Программирование CD/DVD-приводов в LINUX

Содержание

14. Информационные ресурсы

Введение

Вашему вниманию предлагается черновой вариант электронной книги, в которой рассматриваются некоторые вопросы программирования устройств чтения/записи CD/DVD дисков в операционной системе Linux. В частности, в книге рассмотрены порядок управления приводом с использованием регистров контроллера, логическая структура оптических носителей информации, примеры использования встроенных в ядро операционной системы ATAPI- и SCSI-драйверов, алгоритмы записи различной информации на CD и DVD-R/-RW диски (музыкальных треков и данных) и программная реализация данных алгоритмов.

Практически все материалы, находящиеся в данном документе, ранее были опубликованы в журнале «Системный администратор», www.samag.ru.

Работоспособность всех примеров программ была проверена для ОС Linux, ядра 2.4.26/28/31. Привод CD-ROM подключен как Secondary Slave, в ядре включены режим SCSI эмуляции для ATAPI-устройств (SCSI host adapter emulation for IDE ATAPI devices) и поддержка SCSI Generic драйвера. Использовались следующие модели CD/DVD-приводов:

- ASUS DRW-1604P 1.09
- MITSUMI CD-ROM FX54++M Y01E
- MITSUMI CR-48XATE
- _NEC CD-RW NR-9400A R800
- _NEC DVD-RW ND-3520A
- TEAC CD-W524E 1.0E

1. Общая характеристика интерфейса ATAPI

Как пишет в [7] М.Гук: «Интерфейс ATA - AT Attachment for Disk Drives - разрабатывался в 1986-1990 гг. для подключения накопителей на жестких магнитных дисках к компьютерам IBM PC AT с шиной ISA. Стандарт, разработанный комитетом X3T10, определяет набор регистров и назначение сигналов 40-контактного интерфейсного разъема. Интерфейс появился в результате переноса контроллера жесткого диска ближе к накопителю, на плату электроники с сохранением регистровой модели, т.е. создания устройств со встроенным контроллером - IDE (Integrated Device Electronic). Для подключения к интерфейсу ATA накопителей CD-ROM набора регистров и системы команд ATA недостаточно. Для них существует аппаратно-программный интерфейс ATAPI (ATA Package Interface - пакетный интерфейс ATA). Устройство ATAPI поддерживает минимальный набор команд ATA, который неограниченно расширяется 12-байтным командным пакетом, посылаемым хост-контроллером в регистр данных устройства по команде PACKET. Структура командного пакета пришла от SCSI, что обеспечивает схожесть драйверов для устройств со SCSI и ATAPI».

2. Доступ к ATAPI-устройству через регистры контроллера

Регистры ATAPI-контроллера перечислены в таблице 1:

Таблица 1. Регистры ATAPI-контроллера

Адрес регистра		Назначение регистра	
Канал 1	Канал 2	Режим чтения	Режим записи
0x1F0	0x170	Данные (DR)	
0x1F1	0x171	Ошибка (ER)	Регистр свойств (FR)
0x1F2	0x172	Причина прерывания (IR)	
0x1F3	0x173	Не используется	
0x1F4	0x174	Младший байт счетчика байтов (CL)	
0x1F5	0x175	Старший байт счетчика байтов (CH)	
0x1F6	0x176	Выбор устройства (DS)	
0x1F7	0x177	Состояние (SR)	Команда (CR)

Регистр данных (DR) используется так же, как и регистр данных ATA.

Регистр ошибки (ER) определяет состояние контроллера устройства ATAPI после выполнения операции и доступен только для чтения. Назначение разрядов регистра следующее:
- бит 0 (ILI) - недопустимая длина командного пакета или блока данных;
- бит 1 (EOM) - обнаружен конец дорожки на носителе;
- бит 2 (ABRT) - аварийное прекращение выполнения команды;
- бит 3 - не используется;
- бит 4-7 (Sense Key) - код состояния устройства.

Регистры младшего байта и старшего байта счетчика байтов (CL и CH) используются в режиме PIO и доступны как для чтения, так и для записи информации. Значение счетчика должно быть загружено в эти регистры до того, как код команды будет записан в регистр команд. Значение счетчика должно соответствовать объему передаваемых данных.

В регистре выбора устройства (DS) используется только бит 4 (DEV), с помощью которого осуществляется выбор устройства. Биты 7 и 5 должны иметь значение 1 с целью сохранения совместимости с устаревшими устройствами.

Регистр состояния (SR) отображает состояние устройства. Назначение разрядов регистра следующее:
- бит 0 (CHK) - признак возникновения исключительной ситуации, в регистре ошибки (ER) находится код ошибки;
- биты 1 и 2 игнорируются при считывании информации из регистра;
- бит 3 (DRQ) - признак готовности устройства к обмену данными;
- бит 4 (SERV) - признак готовности к обслуживанию следующей команды (имеет значение только при работе в режиме перекрытия команд);
- бит 5 (DMRD/DF) - признак готовности к передаче в режиме DMA (при CHK = 0) или признак неисправности устройства (при CHK = 1).

Регистр команд (CR) используется для загрузки кода выполняемой команды и 12-байтного командного пакета при выполнении пакетной команды.

Рассмотрим порядок выполнения двух классов пакетных команд – команд, не требующих передачи данных и команд, требующих передачи данных от устройства к хосту в режиме PIO.

2.1 Пакетные команды, не требующие передачи данных

Алгоритм выполнения пакетных команд, не требующих передачи данных (Non-data Commands) следующий (см. п. 5.13, спецификация INF-8020i):
- хост считывает регистра состояния устройства, дожидаясь нулевого значения битов BSY и DRQ. После этого хост заносит в регистр выбора устройство байт, бит DEV которого указывает на адресуемое устройство;
- хост записывает код пакетной команды 0xA0 в командный регистр;
- устройство устанавливает бит BSY в регистре состояния и готовится к приему пакетной команды;
- подготовившись к приему пакетной команды, устройство сбрасывает бит BSY и устанавливает бит DRQ;
- хост записывает 12-ти байтный командный пакет в регистр данных устройства;
- устройство устанавливает бит BSY и приступает к выполнению поступившей команды. После выполнения команды устройство сбрасывает бит BSY, DRQ и устанавливает бит DRDY;
- хост считывает регистр состояния.

Рассмотрим функцию, которая реализует данный алгоритм. Вначале введем несколько макроопределений:

- биты регистра состояния SR ATAPI-контроллера:

```
#define BSY 0x80
#define DRQ 0x08
#define DRDY 0x40
#define ERR 0x01
```

- значение бита выбора устройства DEV:

```
#define DEV 1 // 0 - Master, 1 - Slave
```

Макроопределения для работы с портами ввода/вывода:

```
#define OUT_P_B(val,port) asm("outb %%al, %%dx"::"a"(val),"d"(port))
#define OUT_P_W(val,port) asm("outw %%ax, %%dx"::"a"(val),"d"(port))
```

```
#define IN_P_B(val,port) asm("inb %%dx, %%al":"=a"(val):"d"(port))
#define IN_P_W(val,port) asm("inw %%dx, %%ax":"=a"(val):"d"(port))
```

Функция send_packet_command реализует алгоритм выполнения пакетных команд, не требующих передачи данных. Входные параметры функции - указатель на 12-байтный командный пакет. Листинг функции находится в файле SOURCE/RAW/LIB/send_cmd.c:

```
void send_packet_command(__u8 *cmd_buff)
{
    int i;
    __u8 status = 0;
    __u16 port, a;

    port = 0x177; // адрес регистра состояния SR

/* В соответствии с алгоритмом, ждем нулевого значения битов BSY и DRQ */
    for(;;) {
        do {
            IN_P_B(status, port);
        } while(status & BSY);
        if(!(status & DRQ)) break;
    }

/* Выбираем устройство, указав значение бита DEV в регистре выбора устройства */
    port = 0x176; OUT_P_B(0xA0 | (DEV << 4), port);

/* В регистр команд записываем код команды PACKET */
    port = 0x177; OUT_P_B(0xA0, port);

/* Ждем сброса бита BSY и установки бита DRQ */
    for(;;) {
        do {
            IN_P_B(status, port);
        } while(status & BSY);
        if(status & DRQ) break;
    }

/* Записываем в регистр данных переданный 12-байтный командный пакет (отметим
 * одну особенность - если записывать по одному байту, команда работать не будет)
 */
    port = 0x170;
    for(i = 0; i < 12; i += 2) {
        *((__u8 *)&a) = cmd_buff[i];
        *((__u8 *)&a + 1) = cmd_buff[i + 1];
        OUT_P_W(a, port);
    }

/* Ждем сброса бита BSY и установки бита DRDY */
    port = 0x177;
    for(;;) {
        do {
            IN_P_B(status, port);
        } while(status & BSY);
        if(status & DRDY) break;
    }
    return;
}
```

В качестве примера выполнения пакетной команды, не требующей передачи данных, рассмотрим команду START/STOP UNIT. По этой команде устройство выполняет операцию открытия/закрытия лотка CD-привода. Формат пакетной команды START/STOP UNIT представлен на рис.1 (см. [3]):

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (1Bh)							
1	Reserved							Immed
2-3	Reserved							
4							LoEj	Start
5-11	Reserved							

Рис.1. Формат команды START/STOP UNIT

Поле Operation code содержит код команды 0x1B, а тип требуемой операции задают бит LoEj и Start:

LoEj	Start	Действие
1	0	открыть лоток CD-ROM
1	1	закрыть лоток CD-ROM

Перед отправкой устройству пакетной команды необходимо проверить его готовность. Проверка готовности выполняется путем посылки устройству команды TEST UNIT READY:

```
void test_unit_ready()
{
    unsigned char cmd_buff[12];
    memset((void *)cmd_buff, 0, 12);
    cmd_buff[0] = 0x00; // код команды TEST UNIT READY
    send_packet_command(cmd_buff);
}
```

Функция открытия/закрытия лотка CD-привода:

```
void open_close(unsigned char flag)
{
    __u8 cmd_buff[12];

    memset((void *)cmd_buff, 0, 12);
    cmd_buff[0] = 0x1B; // код команды START/STOP UNIT
    cmd_buff[4] = flag; // значение бит LoEj / Start

    send_packet_command(cmd_buff);
}
```

Главная функция:

```
int main()
{
    unsigned char flag;
/* Запросим у операционной системы разрешение на прямой
 * доступ к регистрам ATAPI-контроллера
 */
    ioperm(0x170, 8, 1);

/* Ждем готовность устройства и открываем лоток */
    flag = 2;
    test_unit_ready();
    open_close(flag);

    printf("CD-ROM открыт. Нажмите 'Enter' для закрытия.");
    getchar();

/* Закрываем лоток */
    flag = 3;
    test_unit_ready();
    open_close(flag);

    printf("OK. CD-ROM закрыт.\n");

    ioperm(0x170, 8, 0);
    return 0;
}
```

Полный текст программы открытия/закрытия лотка CD-привода находится в файле SOURCE/RAW/OPEN_CLOSE/open_close.c.

Второй пример пакетной команды, не требующей передачи данных от устройства – команда PLAY AUDIO. По этой команде устройство начнет воспроизведение звукового фрагмента с Аудио-CD. Формат команды PLAY AUDIO приведен на рис.2.

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (45h)							
1	Reserved							
2	MSB							
3	Starting Logical Block Address							
4								
5								LSB
6	Reserved							
7	MSB							
8	Transfer Length							LSB
9-11	Reserved							

Рис.2. Формат команды PLAY AUDIO

Поле Starting Logical Block Address содержит логический номер сектора, с которого начинается воспроизведение. Байты номера сектора располагаются в обратном порядке, на что указывают аббревиатуры MSB (Most significant bit) и LSB (Least significant bit). Поле Transfer Length содержит длину воспроизводимого фрагмента в логических секторах.

Следующая функция формирует и посылает устройству пакетную команду PLAY AUDIO (полный листинг находится в файле SOURCE/RAW/PLAY_CD/play_cd.c):

```
void play_audio()
{
    __u8 cmd_buff[12];

/* Воспроизводим фрагмент размером 5000 секторов, начиная с 100 - го сектора */
    __u32 start_lba = 100;
    __u16 lba_len = 5000;

    memset((void *)cmd_buff, 0, 12);
    cmd_buff[0] = 0x45; // код команды PLAY AUDIO

/* Меняем порядок следования байт в поле Starting Logical Block Address  при помощи макроса __swab32. Этот
 * макрос определен в файле <linux/byteorder/swab.h>
 */
    start_lba = __swab32(start_lba);
    memcpy((void *)(cmd_buff + 2), (void *)&start_lba, 4);

/* Заполняем поле Transfer Length */
    cmd_buff[7] = *((__u8 *)&lba_len + 1);
    cmd_buff[8] = *(__u8 *)&lba_len;

/* Посылаем устройству команду */
    send_packet_command(cmd_buff);
}
```

2.2 Пакетные команды, требующие передачи данных от устройства к хосту в режиме PIO

Следующий класс пакетных команд, который мы рассмотрим, требуют передачи данных от устройства к хосту в режиме PIO. Алгоритм выполнения таких команд следующий (см. п.5.8, спецификация INF-8020i):

- хост считывает регистра состояния устройства, дожидаясь нулевого значения битов BSY и DRQ. После этого хост заносит в регистр выбора устройство байт, бит DEV которого указывает на адресуемое устройство, в регистр счетчика байтов (старший и младший) заносится число передаваемых от устройства байт.
- хост записывает код пакетной команды 0xA0 в командный регистр;
- устройство устанавливает бит BSY в регистре состояния и готовится к приему пакетной команды;
- подготовившись к приему пакетной команды, устройство сбрасывает бит BSY и устанавливает бит DRQ в регистре состояния;
- хост записывает 12-ти байтный командный пакет в регистр данных устройства;

- устройство устанавливает бит BSY и приступает к выполнению поступившей команды. После выполнения команды устройство сбрасывает бит BSY и устанавливает бит DRQ. Если во время выполнения команды произошла ошибка, в регистре состояния будет установлен бит CHK;
- хост опрашивает регистр состояния, дожидаясь единичного значения бита DRQ. После этого хост считывает из регистра данных результат выполнения команды.

Реализуем этот алгоритм при помощи функции send_packet_data_command. Параметры функции - размер запрашиваемых данных и указатель на 12-байтный командный пакет. Листинг функции находится в файле SOURCE/RAW/LIB/send_cmd.c:

```
int send_packet_data_command(__u16 data_len, __u8 *cmd_buff)
{
    int i;
    __u8 status = 0;
    __u16 port, a;

/* Ожидаем сброса битов BSY и DRQ */
    port = 0x177;
    for(;;) {
        do {
            IN_P_B(status, port);
        } while(status & BSY);
        if(!(status & DRQ)) break;
    }

/* В младший и старший байты счетчика байтов (CL и CH) заносим  размер запрашиваемых данных */
    port = 0x174; OUT_P_B(*((__u8 *)&data_len), port);
    port = 0x175; OUT_P_B(*((__u8 *)&data_len + 1), port);

/* Выбираем устройство, указав значение бита DEV в регистре выбора устройства */
    port = 0x176; OUT_P_B(0xA0 | (DEV << 4), port);

/* В  регистр команд записываем код  команды PACKET  */
    port = 0x177; OUT_P_B(0xA0, port);

/* Ждем сброса бита BSY и  установки бита DRQ */
    for(;;) {
        do {
            IN_P_B(status, port);
        } while(status & BSY);
        if(status & DRQ) break;
    }

/* В регистр данных записываем 12-байтный командный пакет */
    port = 0x170;
    for(i = 0; i < 12; i += 2) {
        *((__u8 *)&a) = cmd_buff[i];
        *((__u8 *)&a + 1) = cmd_buff[i + 1];
        OUT_P_W(a, port);
    }

/* Ждем завершения команды - установленного бита DRQ.
 * Если произошла ошибка (установлен бит ERR)  - фиксируем этот факт
 */
    port = 0x177;
    for(;;) {
        do {
            IN_P_B(status, port);
        } while(status & BSY);

/*Фиксируем  ошибку выполнения команды */
        if(status & ERR) return -1;
        if(status & DRQ) break;
    }
    return 0;
}
```

В качестве примера пакетной команды, требующей передачи данных от устройства к хосту, рассмотрим команду INQUIRY. По этой команде устройство выдает блок информации о своих параметрах. Формат команды INQUIRY приведен на рис.3:

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (12h)							
1-3	Reserved							
4	Allocation Length							
5-11	Reserved							

Рис.3. Формат команды INQUIRY

Байт 0 содержит код команды INQUIRY, а байт 4 - размер выделенной области для считываемых данных.

Формат стандартной справки, выдаваемой по команде INQUIRY, приведен на рис.4 (INF-8020i, п. 10.8.1.1, стр.94.):

Bit / Byte	7	6	5	4	3	2	1	0
0	Reserved			Peripheral Device Type				
1	RMB	Reserved						
2	ISO Version		ECMA Version			ANSI Version (00)		
3	ATAPI Version				Response Data Format			
4	Additional Length (Number of bytes following this one)							
5-7	Reserved							
8-15	Vendor Identification							
16-31	Product Identification							
32-35	Product Revision Level							
36-…	Vendor-specific							

Рис.4. Формат стандартной справки, выдаваемой по команде INQUIRY

Функция read_inquiry() считывает идентификатор изготовителя (Vendor Identification), идентификатор изделия (Product Identification) и версию изделия (Product Revision Level):

```
int read_inquiry()
{
    int i = 0;
    __u8 cmd_buff[12];
    __u16 inquiry_buff[20]; // буфер для считываемых данных

    memset((void *)inquiry_buff, 0, 40);
    memset((void *)cmd_buff, 0, 12);
    cmd_buff[0] = 0x12; // код команды INQUIRY
    cmd_buff[4] = 40; // размер считываемых данных – 40 байт

/* Посылаем устройству пакетную команду */
    if(send_packet_data_command(40, cmd_buff) < 0) {
        printf("INQUIRY error\n");
        return -1;
    }

/* Считываем и отображаем результат */
    for(i = 0; i < 20; i++) IN_P_W(inquiry_buff[i], 0x170);
    printf("%.8s%.16s%.4s\n", inquiry_buff + 4, \
                inquiry_buff + 8, inquiry_buff + 16);
    return 0;
}
```

Полный листинг программы чтения идентификационной информации устройства находится в файле SOURCE/RAW/INQUIRY/inquiry_cd.c

Перед тем как рассмотреть примеры выполнения других пакетных команд этого класса, необходимо ознакомится с организацией данных компакт-диске.

3. Организация данных на компакт-диске

Пространство CD-ROM делится на три области:
- Lead-In Area, входная область диска
- User Data Area, область данных пользователя, или область программ
- Lead-Out Area, выходная область диска

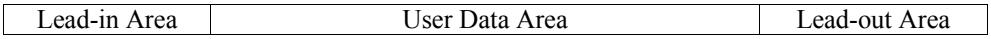

Рис.5. Общая структура CD-ROM

Lead-In область предназначена для хранения информации об организации данных на компакт-диске и содержит таблицу содержания диска, Table Of Contents (TOC).

В User Data Area находятся треки с данными пользователя. Минимальное число треков, которое может быть записано на компакт-диск, равно 1. Максимальное число треков на диске равно 99. Треки нумеруются, начиная с единицы.

Совокупность всех трех областей - Lead-In, User Data, Lead-Out - называется сессией. Любой CD-ROM, содержащий информацию, имеет минимум одну сессию. Максимальное число сессий зависит от используемого типа компакт-диска. Нумерация сессий начинается с единицы.

На рис.6 представлена общая структура многосессионного компакт-диска.

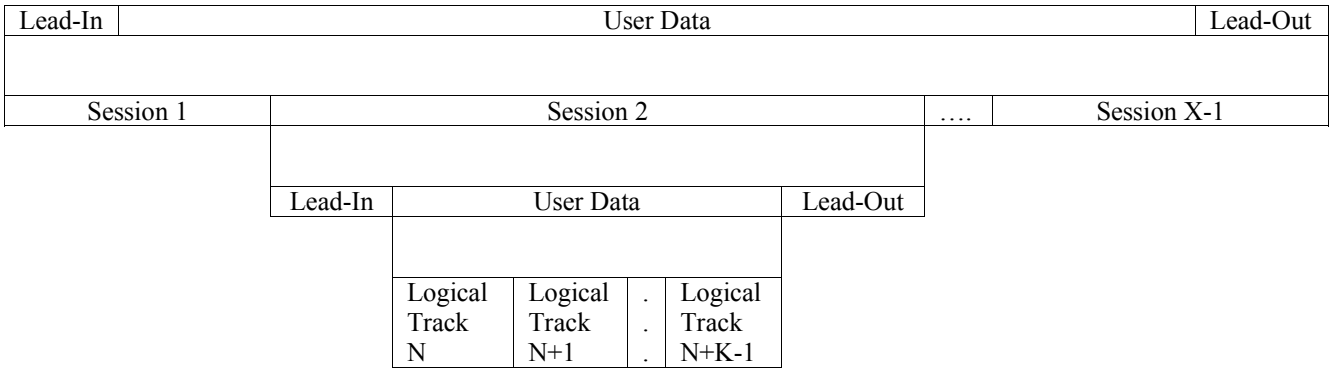

Рис.6. Общая структура многосессионного компакт-диска

Lead-In область первой сессии является Lead-In областью всего диска. Lead-Out область последней сессии является Lead-Out областью диска. В User Data области любой сессии находятся треки с данными.

Единицей представления данных на компакт-диске является малый кадр, small frame. Малый кадр содержит:
- 3 байта кода синхронизации;
- 1 байт данных субканала;
- 24 байта данных основного канала - две группы по 12 байт, защищенных помехоустойчивым корректирующим кодом Рида-Соломона (CIRC, Cross Interleaved Reed-Solomon Code).

При записи на компакт-диск данные субканала, основного канала и CIRC кодируются 14-разрядными EFM-кодом (Eight to Fourteen Modulation). Эта операция необходима для соблюдения условия, что в последовательном коде данных между двумя соседними единицами должно быть не более 10, но не менее 2 нулей. Дополнительно к каждому полю добавляются три связывающих бита. Итоговый размер малого кадра, записанного на компакт-диск, равен 588 бит (рис.7).

98 последовательно расположенных малых кадров образуют кадр (Frame) - минимально адресуемую единицу данных на компакт-диске. Один кадр содержит 24 x 98 = 2352 байт данных основного канала и 98 байт субканала. Порядок формирования блока данных основного канала кадра показан на рис.8.

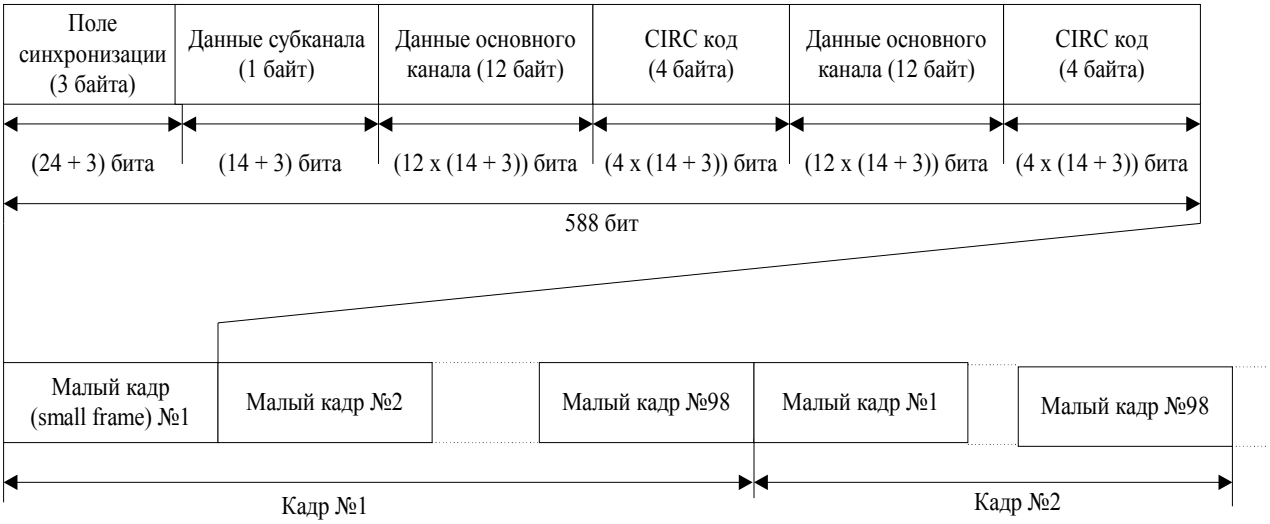

Рис.7. Структура кадра компакт-диска

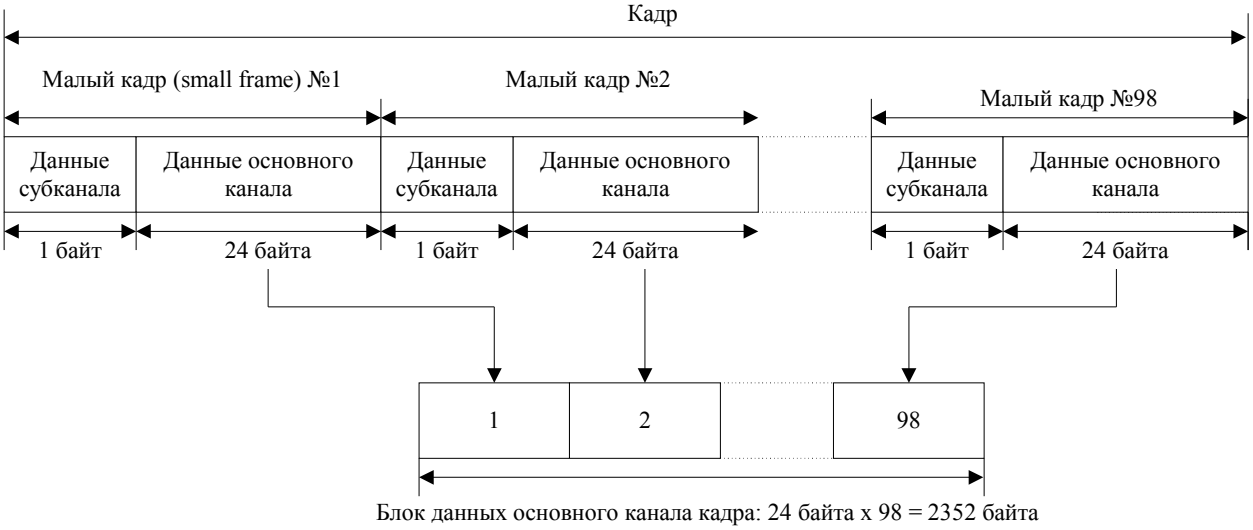

Рис.8. Порядок формирования блока данных основного канала кадра

Блок данных основного канала предназначен для хранения информации пользователя - аудио или данных. У компакт-дисков, используемых для хранения аудио-информации, все 2352 байт блока основного канала заняты аудиоданными. Для хранения данных используется 6 основных форматов блока основного канала. Наиболее широкое применение получили три формата:

- режим данных 1, Yellow Book Mode 1
- форма 1 режима данных 2, CD-ROM XA Mode 2 Form 1
- форма 2 режима данных 2, CD-ROM XA Mode 2 Form 2

Блок данных основного канала, содержащий данные, начинается с поля синхронизации Data Block Sync Pattern. Размер этого поля 12 байт (рис.9). За полем синхронизации находится заголовок блока Header данных длиной 4 байт. Заголовок блока содержит координаты блока данных в формате MSF (Minute/Second/Frame) и байт формата записи данных Data Mode.

0	1	2	3	4	5	6	7	8	9	10	11
00h	FFh	FFh	FFh	FFh	FFh	FFh	FFh	FFh	FFh	FFh	00h

Рис.9. Структура поля синхронизации Data Block Sync Pattern

Формат данных Mode 1 представлен в таблице 2, форматы данных Mode 2 Form 1 и Mode 2 Form 2 - в таблицах 3 и 4.

Таблица 2. Формат данных Mode 1

Смещение (в байтах)	Размер поля (в байтах)	Содержание
0	12	Поле синхронизации (Data Block Sync pattern)
12	3	Адрес блока в формате MSF, в BCD коде
15	1	Режим данных, Data Mode = 01b
16	2048	Данные пользователя, User Data
2064	4	CRC (P = (X16+X15+X2+X1)*(X16+X2+X+1)) Bytes 0 - 2063
2068	8	Заполнено нулями
2046	172	P parity symbols
2248	104	Q parity symbols

Таблица 3. Формат данных Mode 2 Form 1

Смещение (в байтах)	Размер поля (в байтах)	Содержание
0	12	Поле синхронизации (Data Block Sync pattern)
12	3	Адрес блока в формате MSF, в BCD коде
15	1	Режим данных Data Mode = 10b
16	4	Первая копия подзаголовка (Sub-header, first copy)
20	4	Вторая копия подзаголовок (Sub-header, second copy)
24	2048	Данные пользователя, User data
2072	4	CRC (P = (X16+X15+X2+X1)*(X16+X2+X+1)), Bytes 16 - 2071
2076	172	P parity symbols
2248	104	Q parity symbols

Таблица 4. Формат данных Mode 2 Form 2

Смещение (в байтах)	Размер поля (в байтах)	Содержание
0	12	Поле синхронизации (Data Block Sync pattern)
12	3	Адрес блока в формате MSF, в BCD коде
15	1	Режим данных Data mode = 2
16	4	Первая копия подзаголовка (Sub-header, first copy)
20	4	Вторая копия подзаголовок (Sub-header, second copy)
24	2324	Данные пользователя, User data
2348	4	Optional CRC Bytes 16 - 2347

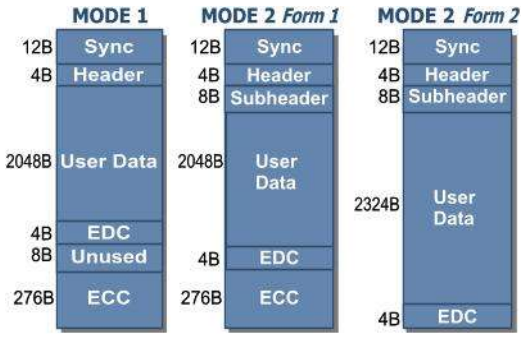

Рис.10. Форматы блока данных основного канала

98 байт субканала делятся на 2 байта синхронизации и 96 байт данных. Каждый байт данных субканала размечен на битовые позиции, и, таким образом, субканал делится еще на 8 субканалов. Формат байта данных субканала представлен на рис.11:

7	6	5	4	3	2	1	0
P	Q	R	S	T	U	V	W

Рис.11. Формат байта данных субканала

Порядок формирования блока данных субканала на примере Q-субканала показан на рис.12.

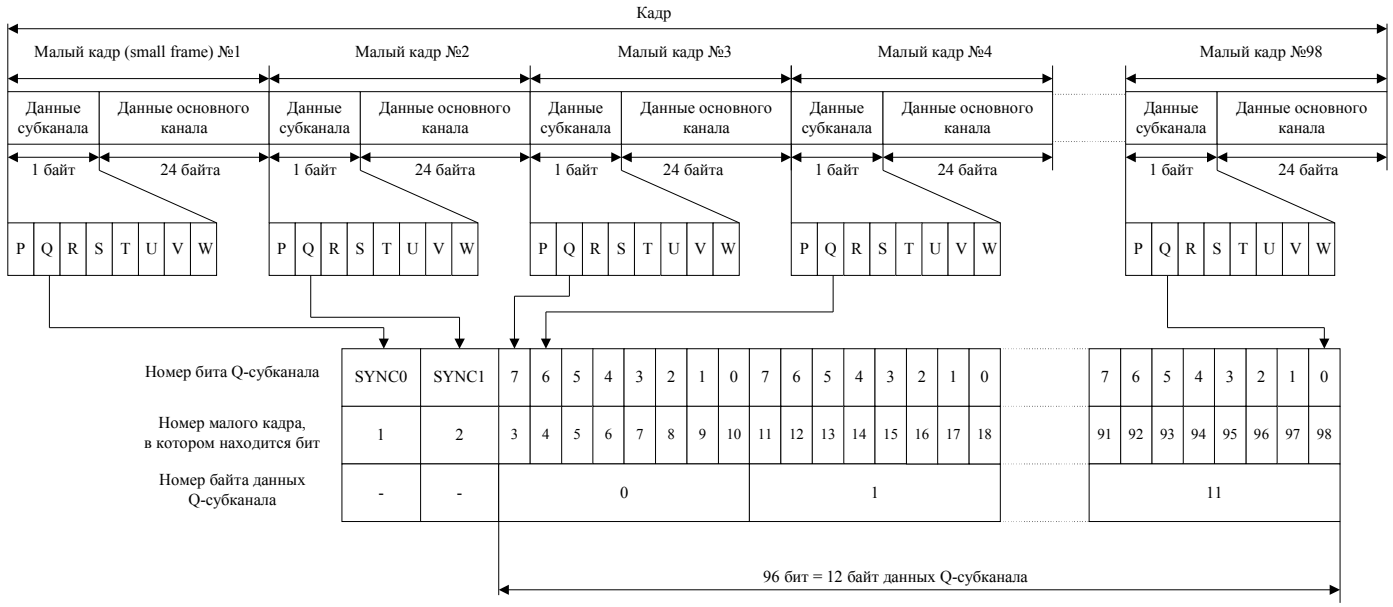

Рис.12. Порядок формирования блока данных субканала (на примере Q-субканала)

Из всех имеющихся в наличии субканалов основную информационную нагрузку несет именно Q-субканал. Характер данных, находящихся в Q-субканале, зависит от места расположения субканала на диске. Так, Q-субканал области данных содержит адресную информацию кадра, Q-субканал Lead-In области хранит информацию об организации данных на компакт-диске, таблицу содержания диска. Рассмотрим общий формат данных Q-субканала. Q-субканал содержит:

- 2 бита синхронизации (S0, S1);
- 4 бита поля управления (Control Field);
- 4 бита идентификатора режима данных Q-субканала (ADR);
- 72 бита (9 байт) данных (DATA-Q);
- 16 бит контрольной суммы полей управления, режима данных и поля данных (CRC).

Поле Control Field определяет тип информации, которая находится в основном канале кадра. Это поле может принимать следующие значения:

- 00xxb - 2 аудиоканала
- 10xxb - 4 аудиоканала
- 01xxb - трек данных
- 11xxb - зарезервировано

Поле ADR определяет режим записи данных Q-субканала. Основным режимом записи Q-субканала области данных пользователя (User Data Area) является Mode-1 Q. В этом режиме поле ADR = 1. Формат данных Q-субканала User Data Area, режим Mode-1 Q, представлен на рис.13.

ADR	DATA-Q								
0001	TNO	INDEX	MIN	SEC	FRAME	ZERO=00	AMIN	ASEC	AFRAME

Рис.13. Формат данных Q-субканала User Data Area, режим Mode-1 Q

Назначение полей:

- TNO – номер текущего трека
- INDEX – индекс трека
- MIN, SEC, FRAME – координаты текущего кадра относительно начала трека
- ZERO – разделительное поле, содержит 0
- AMIN, ASEC, AFRAME – абсолютные координаты текущего кадра (относительно начала диска)

Все координаты представлены в BCD коде.

Структура Q-субканала Lead-Out области совпадает со структурой Q-субканала области данных. В режиме Mode-1 Q поле TNO содержит значение 0xAA, поле INDEX = 01bcd.

Q-субканал Lead-In области хранит таблицу содержания диска, Table Of Contents (TOC). Формат данных Q-субканала Lead-In области представлен на рис.14.

ADR	DATA-Q								
0001/005	TNO=00	POINT	MIN	SEC	FRAME	ZERO=00	AMIN	ASEC	AFRAME

Рис.14. Формат данных Q-субканала Lead-In области

Данные Q-субканала Lead-In области записываются в двух режимах – Mode-1 Q (ADR = 1) и Mode-5 Q (ADR = 5). В зависимости от режима, поле POINT определяет назначение остальных полей (см. табл.5).

Таблица 5. Область Lead-In, формат Q-субканала

ADR	POINT	MIN	SEC	FRAME	PMIN	PSEC	PFRAME
1	01-99	---			Стартовая позиция трека, номер трека указан в поле POINT		
1	A0	---			Номер первого трека сессии	Тип диска	00
1	A1	---			Номер последнего трека сессии	00	00
1	A2	---			Стартовые координаты Lead-Out области		
5	B0	Стартовые координаты следующей возможной области программ			Максимально возможные стартовые координаты Lead-Out области		
5	C0	---			Стартовые координаты Lead-In области диска		

Диски CD-R и CD-RW содержат две дополнительные области перед первой Lead-In областью компакт-диска - Power Calibration Area (PCA) и Program Memory Area (PMA). Структура CD-R/RW диска представлена на рис.15.

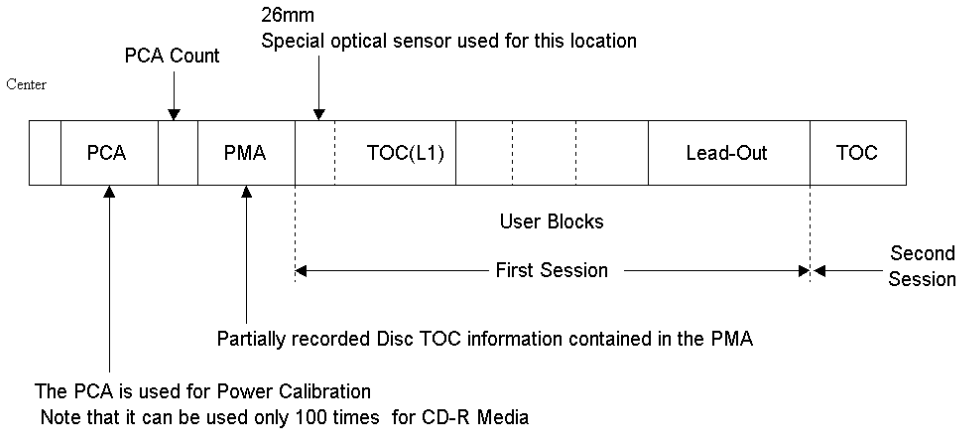

Рис.15. Структура CD-R и CD-RW диска

The Power Calibration Area, PCA - область калибровки мощности пишущего лазера. Область присутствует только на CD-R и CD-RW дисках. В свою очередь, PCA делится на две области - тестовую область (test area) и область счетчика (count area). Тестовая область содержит 100 калибровочных участков. Область счетчика используется для учета количества использованных калибровочных участков.

The Program Memory Area, PMA - область памяти программ. Область присутствует только на CD-R и CD-RW дисках и предназначена для учета использования User Data области носителя.

Q-субканал области PMA используется в качестве временной TOC при записи треков в режимах TAO (Track-at-once) и SAO (Session-at-once). При записи диска в режиме DAO (Disk-at-once) данные в PMA не записываются.

Подробное описание форматов данных Q-субканала приведено в спецификации SCSI MMC-5 ([1]), п. 4.3.4.4 "Q Sub-channel". Ниже будут рассмотрены примеры чтения данных Q-субканала различных областей компакт-диска.

4. Примеры выполнения пакетных команд, требующих передачи данных от устройства к хосту в режиме PIO

4.1 Чтение сектора с компакт-диска

Для чтения секторов с компакт-диска используется команда READ CD. Формат этой команды представлен на рис.16.

<table>
<tr><td>Bit
Byte</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td colspan="8">Operation code (BEh)</td></tr>
<tr><td>1</td><td colspan="4">Reserved</td><td colspan="2">Expected Sector Type</td><td>DAP</td><td>RelAdr</td></tr>
<tr><td>2</td><td colspan="8" rowspan="4">MSB
Starting Logical Block Address
LSB</td></tr>
<tr><td>3</td></tr>
<tr><td>4</td></tr>
<tr><td>5</td></tr>
<tr><td>6</td><td colspan="8" rowspan="3">MSB
Transfer Length
LSB</td></tr>
<tr><td>7</td></tr>
<tr><td>8</td></tr>
<tr><td rowspan="2">9</td><td colspan="5">Main Channel Selection Bits</td><td colspan="2" rowspan="2">C2 Error Information</td><td rowspan="2">Reserved</td></tr>
<tr><td>SYNC</td><td>Header Codes</td><td>User Data</td><td>EDC & ECC</td></tr>
<tr><td>10</td><td colspan="6">Reserved</td><td colspan="2">Sub-channel Selection Bits</td></tr>
<tr><td>11</td><td colspan="8">Reserved</td></tr>
</table>

Рис.16. Формат команды READ CD

Назначение полей командного пакета:

- Expected Sector Type - определяет тип сектора, который мы хотим считать. Может принимать следующие значения (INF-8090i, таблица 289):
  - 000b – сектор любого типа (any type)
  - 001b - CD-DA
  - 010b - Mode1
  - 100b - Mode2 Form1
  - 101b - Mode2 Form2
- Starting Logical Block Address - номер стартового сектора;
- Transfer Length in Blocks - число считываемых секторов;
- Main Channel Selection Bits - это поле определяет, какие данные будут прочитаны из сектора. В таблице 294 (стр. 397) спецификации INF-8090i, определены допустимые значения этого поля, и поля данных, которые будут считаны из сектора (секторов). Например, если девятый байт содержит значение 0xF8, то из сектора любого типа (Mode1, Mode 2 Form 1/2) будут считаны все данные: поле синхронизации Sync, заголовки Header и Subheader (если используется Mode 2), поле данных и значения контрольных сумм EDD/ECC. Другими словами, с компакт-диска будет прочитан "сырой" сектор (RAW-сектор);
- Sub-Channel Data Selection Bits - данное поле определяет необходимость считывания в общем потоке данных содержимого субканалов:
  - 000b - данные из субканалов не считываются;
  - 001b - считываются "сырые" данные субканалов;
  - 010b - данные Q-субканала;
  - 100b - данные R-W субканалов.

Рассмотрим функцию, которая выполняет чтение RAW-сектора с компакт-диска. Входные данные функции - логический номер сектора, содержимое которого мы хотим прочитать.

```
int read_cd(__u32 lba)
{
    int i = 0, out_f;
    __u8 cmd_buff[12];
    __u8 buff[CD_FRAMESIZE_RAW];
    __u16 a = 0;

    memset((void *)buff, 0, sizeof(buff));
    memset((void *)cmd_buff, 0, 12);
```

/ Формируем командный пакет (см. рис.16) */*

```
    cmd_buff[0] = 0xBE; // код команды READ CD
    cmd_buff[1] = 0; // считываем сектор любого типа (Any Type)
    cmd_buff[9] = 0xF8; // считываем «сырой» сектор(RAW-сектор)
    cmd_buff[8] = 1; // считываем один сектор

/* Заполняем поле Starting Logical Block Address, при этом
 * меняем порядок расположения байт
 */
    printf("LBA: %u\n", lba);
    lba = __swab32(lba);
    memcpy((cmd_buff + 2), (void *)&lba, 4);

/* Посылаем устройству командный пакет */
    if(send_packet_data_command(CD_FRAMESIZE_RAW, cmd_buff) < 0) {
        request_sense();
        return -1;
    }

/* Считываем результат и сохраняем его в файле */
    for(i = 0; i < CD_FRAMESIZE_RAW; i += 2) {
        IN_P_W(a, 0x170);
        buff[i] = *((__u8 *)&a);
        buff[i + 1] = *((__u8 *)&a + 1);
    }

    out_f = open("sector", O_CREAT|O_RDWR, 0600);
    write(out_f, buff, sizeof(buff));

    return 0;
}
```

Если при обращении к устройству произойдет ошибка, то никаких данных мы, естественно, не получим. Однако всегда есть возможность узнать причину ошибки. Для этого достаточно послать устройству пакетную команду REQUEST SENSE. Формат этой команды приведен на рис.17.

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (03h)							
1-3	Reserved							
4	Allocation Length							
5-11	Reserved							

Рис.17. Формат команды REQUEST SENSE

В ответ на команду REQUEST SENSE устройство выдаст информационный блок SENSE DATA следующего формата:

Bit / Byte	7	6	5	4	3	2	1	0
0	Valid	Error Code (70h or 71h)						
1	Segment Number (Reserved)							
2	Reserved		ILI	Reserved	Sense Key			
3-6	Information							
7	Additional Sense Length (n - 7)							
8-11	Command Specific Information							
12	Additional Sense Code							
13	Additional Sense Code Qualifier (Optional)							
14-n	---							

Рис.18. Блок данных SENSE DATA, возвращаемый по команде REQUEST SENSE

Три поля данной структуры - Sense Key, Additional Sense Code (ASC), Additional Sense Code Qualifier - позволяют точно установить причину ошибки. Допустимые значения этих полей и их описание ошибочной ситуации приведены в спецификации INF-8090i, таблицы 499-504, стр. 547 - 563.

Анализ ошибочной ситуации выполняет функция request_sense():

```
int request_sense()
```

```
{
    int i = 0;
    __u8 cmd_buff[12];
    __u8 sense_buff[14];
    __u16 a;

#define SK (sense_buff[2] & 0x0F)
#define ASC sense_buff[12]
#define ASCQ sense_buff[13]

    memset((void *)cmd_buff, 0, 12);
    memset((void *)sense_buff, 0, 14);

/* Формируем пакетную команду REQUEST SENSE. Из блока sense data считываем первые 14 байт -
 * этого нам хватит, чтобы определить причину ошибки
 */
    cmd_buff[0] = 0x3;
    cmd_buff[4] = 14;

/* Посылаем устройству команду и считываем sense data */
    if(send_packet_data_command(14, cmd_buff) < 0) {
        printf("Error request sense\n");
        exit(-1);
    }

    for(i = 0; i < 14; i += 2) {
        IN_P_W(a, 0x170);
        sense_buff[i] = *(__u8 *)&a;
        sense_buff[i + 1] = *((__u8 *)&a + 1);
    }

    printf("SK/ASC/ASCQ: 0x%X/0x%X/0x%X\n", SK, ASC, ASCQ);
    return 0;
}
```

Полный листинг программы чтения секторов с компакт-диска находится в файле SOURCE/RAW/READ_SECTOR/read_sector.c. Приведем пример работы этой программы. С компакт-диска, содержащего данные, считывается сектор номер 1000. В результате работы будет создан файл sector. Посмотрим на первые 16 байт этого файла:

```
00 FF FF FF FF FF FF FF FF FF FF 00 00 15 25 01
```

Первые 12 байт - это поле синхронизации Sync сектора. Следующие за ним 3 байта - координаты сектора в формате MSF, значения представлены в BCD коде. Последний байт содержит значение режима записи данных, Data Mode.

Пересчитаем координаты сектора из MSF формата в LBA по формуле:

```
LBA = ((Minute * 60 + Second) * 75 + Frame) – 150, (см. SCSI MMC-5 [1])
```

В нашем примере, Minute = 00, Second = 15, Frame = 25. Подставив значения в формулу, получаем LBA = 1000. Именно этот сектор мы считывали.

Теперь проверим, как обрабатываются ошибочные ситуации. Удалим из привода компакт-диск и запустим программу на выполнение. В результате программа выдаст следующее:

```
SK/ASC/ASCQ: 0x2/0x3A/0x0
```

За расшифровкой обратимся к таблицам 499-504. Значение Sense key, равное 0x2, означает "NOT READY. Indicates that the Device cannot be accessed", ASC = 0x3A и ASCQ = 0x0 - "MEDIUM NOT PRESENT". Устройство сообщило о том, что в лотке отсутствует компакт-диск.

Теперь попытаемся прочитать сектора, номер которого заведомо превышает допустимое значение. В программе задаем LBA = 10000000, и получаем в результате:

```
SK/ASC/ASCQ: 0x5/0x21/0x0
```

Расшифровываем полученные значения:
```
0x5 - ILLEGAL REQUEST
0x21 - LOGICAL BLOCK ADDRESS OUT OF RANGE.
```

В этом случае устройство сообщает, что логический адрес сектора вышел за пределы допустимого диапазона.

4.2 Чтение таблицы содержания диска

Чтение таблицы содержания диска (данные Q-субканала Lead-In области) выполняет команда READ TOC/PMA/ATIP. Формат этой команды представлен на рис.19.

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (43h)							
1	LUN (Obsolete)			Reserved			MSF	Reserved
2	Reserved				Format			
3-5	Reserved							
6	Track / Session Number							
7	MSB							
8	Allocation Length							LSB
9-11	Reserved							

Рис.19. Формат команды READ TOC/PMA/ATIP

Назначение полей командного пакета:
MSF – определяет формат адреса блока данных (0 - LBA, 1 - MSF);
Format – определяет формат данных, выдаваемых по команде READ TOC/PMA/ATIP:
- 00b - считываются данные TOC, начиная с трека, номер которого указан в поле Track/Session Number. Если это поле содержит 0, содержимое TOC выдается для всех треков диска, начиная с первого. Если поле Track/ Session Number содержит значение 0xAA, выдаются данные Lead-Out области последней завершенной сессии.
- 01b - считывается номер первой завершенной сессии, номер последней завершенной сессии и номер первого трека последней сессии.
- 10b - считываются все данные Q-субканала Lead-In областей всех сессий, начиная с сессии, номер которой находится в поле Session Number.

Формат данных TOC для Format = 00b, приведен на рис.20.

Bit / Byte	7	6	5	4	3	2	1	0
0	MSB							
1	TOC Data Length							LSB
2	First Track Number							
3	Last Track Number							
	TOC Track Descriptors							
0	Reserved							
1	ADR				Control			
2	Track Number							
3	Reserved							
4	MSB							
5	Absolute CD-ROM Address							
6								
7								LSB

Рис.20. Формат данных TOC (Format Field = 00b)

Первые четыре байта – это заголовок TOC. Поле TOC Data Length содержит размер данных TOC, при этом размер самого поля TOC Data Length (2 байта) не учитывается. Поля First Track Number и Last Track Number содержат номера первого и последнего треков на диске. За заголовком следуют дескрипторы треков. Первым расположен дескриптор трека, номер которого задан в поле Starting Track. Максимальный размер данных TOC, согласно спецификации INF-8020i, составляет 804 байта (100 дескрипторов треков по 8 байт каждый + 4 байта заголовка).

Рассмотрим функцию read_toc, выполняющую чтение TOC при условии, что поле Format содержит 00b. Формат дескриптора трека TOC, представленный на рис.20, описывает структура struct toc:

```
struct toc {
       __u8 res;
       __u8 adr_cntl;
       __u8 trk_num; // номер трека
       __u8 res1;
       __u32 lba; // адрес сектора
};

int read_toc()
{
    int i = 0;
    struct toc *t;
    __u8 cmd_buff[12];
    __u8 *data_buff;
    __u16 a, toc_len = 0; // toc_len - размер TOC

/* Формируем пакетную команду. Поле Track/Session Number содержит 0,
 * по команде READ TOC будет выдана информация обо всех треках диска,
 * начиная с первого
 */
    memset((void *)cmd_buff, 0, 12);
    cmd_buff[0] = 0x43; // код команды READ_TOC
    cmd_buff[6] = 0; // Track/Session Number = 0

/* Заполняем поле Allocation Length. Размер данных TOC заранее
 * нам неизвестен, поэтому запросим максимальное значение – 804 байт
 */
    toc_len = 804;
    cmd_buff[7] = *((__u8 *)&toc_len + 1);
    cmd_buff[8] = *((__u8 *)&toc_len);

/* Выделяем память для хранения данных TOC */
    data_buff = (__u8 *)malloc(804);
    memset(data_buff, 0, 804);

/* Отправляем устройству сформированную пакетную команду */
    if(send_packet_data_command(804, cmd_buff) < 0) {
        request_sense();
        return -1;
    }

/* Считываем результат */
    for(i = 0;;i += 2) {
        IN_P_W(a, 0x170);

/* Первые два байта – это размер данных TOC (поле TOC Data Length) */
        if(i == 0) toc_len = __swab16(a);

/* Учитываем длину самого поля TOC Data Length в общем размере данных */
        toc_len += 2;

/* Заполняем буфер данными TOC */
        if(i > toc_len) break;
        data_buff[i] = *((__u8 *)&a);
        data_buff[i + 1] = *((__u8 *)&a + 1);
    }

/* Номер последнего трека на диске */
    total_tracks = data_buff[3];

    printf("TOC length: %d\n", toc_len);
    printf("First: %d\t", data_buff[2]);
    printf("Last: %d\n", total_tracks);

/* Выделяем память и копируем туда дескрипторы треков */
```

```
    t = (struct toc *)malloc(toc_len);
    memcpy((void *)t, (data_buff + 4), toc_len);

    free(data_buff);

/* Отобразим результаты чтения TOC */
    for(i = 0; i < total_tracks; i++)
        printf("track: %d\tlba: %u\n", (i + 1), __swab32((t + i)->lba));

    return 0;
}
```

Полный листинг программы чтения TOC компакт-диска находится в файле SOURCE/RAW/TOC/read_toc1.c.

Рассмотрим еще один пример чтения TOC. Для хранения содержимого TOC определим односвязный список структур следующего вида:

```
struct toc {
    __u8 res;
    __u8 adr_cntl;
    __u8 trk_num;
    __u8 res1;
    __u32 lba;
    struct toc *next;
}
```

Здесь struct toc *next – указатель на следующий элемент списка. Формировать этот список будет рекурсивная функция read_toc:

```
struct toc * read_toc()
{
    static int i = 1;
    int n;
    __u8 cmd_buff[12];
    __u8 data_buff[12];
    __u16 a, toc_len = 0;
    struct toc *t;

/* Если номер трека превысил 0xAA (прочитали Lead-Out диска), выполнение функции прекращается */
    if(i > 0xAA) return NULL;

/* Выделяем память для дескриптора трека */
    t = (struct toc *)malloc(sizeof(struct toc));

/* Формируем пакетную команду */
    memset((void *)cmd_buff, 0, 12);
    cmd_buff[0] = 0x43; // код команды READ TOC

/* Поле Starting Track содержит номер трека. Этот номер увеличивается
 * при каждом вызове функции
 */
    cmd_buff[6] = i;

/* При каждом обращении к диску мы считываем 12 байт - 4 байта заголовка
 * и 8 байт дескриптора трека, номер которого задан в поле Track/Session Number
 */
    cmd_buff[8] = 12;

/* Посылаем устройству пакетную команду */
    if(send_packet_data_command(12, cmd_buff) < 0) {
        printf("Error read TOC\n");
        request_sense();
        exit(-1);
    }

/* Считываем данные - заголовок и дескриптор трека*/
    memset(data_buff, 0, 12);
    for(n = 0; n < 12; n += 2) {
        IN_P_W(a, 0x170);
```

```
            data_buff[n] = *((__u8 *)&a);
            data_buff[n + 1] = *((__u8 *)&a + 1);
        }

/* Отобразим размер данных TOC и номера первого и последнего трека */
        if (i == 1) {
            *((__u8 *)&toc_len) = data_buff[1];
            *((__u8 *)&toc_len + 1) = data_buff[0];

            printf("TOC lenght - %d\n", toc_len);
            printf("First: %d\t", data_buff[2]);
            printf("Last: %d\n", data_buff[3]);
            max_track_num = data_buff[3];
        }

/* Копируем дескриптор трека в структуру struct toc */
        memcpy((void *)t, (data_buff + 4), 8);
        t->lba = __swab32(t->lba);

/* Считываем дескриптор следующего трека. Если треков больше нет,
 * считываем дескриптор Lead-Out области последней сессии.
 */
        i += 1;
        if(i == (max_track_num + 1)) i = 0xAA;

        t->next = read_toc();
        return t;
}
```

Просмотр содержимого TOC выполняет рекурсивная функция view_toc:

```
void view_toc(struct toc *t)
{
    if(t == NULL) return;

    if(t->trk_num == 0xAA) printf("lead out:\t");
    else printf("track: %d\t", t->trk_num);
    printf("lba: %u\n", t->lba);

    view_toc(t->next);
}
```

Извлечь из сформированного списка дескриптор трека можно при помощи функции get_toc_entry:

```
struct toc_entry * get_toc_entry(int trk_num, struct toc *t)
{
    struct toc_entry *t_entry;
    int i = 1;

    for(;i < trk_num; i++) t = t->next;

    t_entry = (struct toc_entry *)malloc(sizeof(struct toc_entry));
    t_entry->start_lba = t->lba;
    t_entry->end_lba = t->next->lba;

    return t_entry;
}
```

Входные параметры функции - номер трека и указатель на начало списка с данными TOC. Результат сохраняется в структуре `struct toc_entry` следующего вида:

```
struct toc_entry {
       __u32 start_lba; // стартовый адрес трека
       __u32 end_lba; // конечный адрес трека
};
```

Полный листинг программы, выполняющей чтение TOC, приведен в файле SOURCE/RAW/TOC/read_toc.c.

Если скомпоновать вместе функции read_cd и read_toc, можно написать программу чтения треков с компакт-диска. В файле SOURCE/RAW/READ_CDDA/read_cdda_track.c находится листинг программы, которая считывает треки с Audio-CD и сохраняет их в файле track.cdr. Детально изучать этот листинг мы не будем, т.к. только что подробно рассмотрели его основные составляющие. Единственное замечание - в отличии от ранее рассмотренной функции read_cd, при считывании сектора с аудиодиска соседние байты меняются местами. Это связано с порядком расположения аудиоданных в секторе:

```
        for(i = 0; i < 2352; i += 2) {
            IN_P_W(a, 0x170);
            buff[i] = *((__u8 *)&a + 1);
            buff[i + 1] = *(__u8 *)&a;
        }
```

В принципе, этого можно и не делать, но тогда нам придется самостоятельно формировать RIFF-заголовок, чтобы получить файл в формате WAV. Лучше возложим эту почетную обязанность на sox:

```
# sox track.cdr track.wav
```

Полученный WAV-файл можно закодировать в цифровой формат – MP3 или OggVorbis:

```
# oggenc –b 256 track.wav
```

5. Доступ к устройству при помощи файла ATAPI-драйвера

5.1 Файлы устройств и системный вызов IOCTL

Файл - основа любой операционной системы, поскольку именно с ним производится наибольшее число действий. В UNIX- и POSIX-системах существуют файлы следующих типов:

- обычный файл;
- каталог;
- FIFO-файл;
- байт-ориентированный файл устройства;
- блок-ориентированный файл устройства;

Примерами блок-ориентированных устройств являются ATA- и ATAPI-устройства.

Прикладная программа может выполнять операции чтения и записи с файлом устройства так же, как с обычным файлом, а операционная система будет автоматически вызывать соответствующий драйвер устройства для выполнения фактической передачи данных между физическим устройством и приложением.

Файл устройства создается командой mknod, одним из аргументов которой является старший номер устройства (major device number). По сути старший номер - это индекс в таблице ядра, которая содержит адреса всех драйверов, известных системе. В ОС Linux создаются две таблицы - таблица блочных устройств (block device switch) и таблица символьных устройств (character device switch). Обе таблицы являются массивом структур и проиндексированы при помощи значения старшего номера устройства. Таблица блочных устройств определена в файле fs/block_dev.c следующим образом:

```
static struct {
       const char *name;
       struct block_device_operations *bdops;
} blkdevs[MAX_BLKDEV];
```

Этот массив заполняется во время регистрации блочного устройства в системе. Для регистрации устройства соответствующий драйвер вызывает функцию register_blkdev (см. файл fs/block_dev.c):

```
int register_blkdev(unsigned int major, const char * name, \
             struct block_device_operations *bdops)
{
       ....
       blkdevs[major].name = name;
       blkdevs[major].bdops = bdops;
       return 0;
}
```

Аргумент major - старший номер устройства, name - имя файла устройства, структура struct block_device_operations содержит функции, выполняемые драйвером блочного устройства.

При снятии регистрации соответствующий элемент массива blkdevs обнуляется:

```
int unregister_blkdev(unsigned int major, const char * name)
{
	....
	blkdevs[major].name = NULL;
	blkdevs[major].bdops = NULL;
	return 0;
}
```

Когда пользовательский процесс читает данные из файла устройства или записывает их, ядро, используя старший номер устройства в качестве индекса, находит в соответствующей таблице нужную процедуру драйвера и выполняет запрашиваемое действие.

Кроме операций чтения/записи, драйвер также предоставляет возможность управления устройством. Операция управления осуществляется при помощи функции ioctl. Эта функция вызывается пользовательским процессом и имеет следующий прототип:

```
	int ioctl(int fd, int cmd, ...);
```

Аргументы функции:
- int fd - файловый дескриптор устройства;
- int cmd - команда, посылаемая устройству.

Третий параметр является специфичным для каждого устройства, поэтому в прототипе функции не указан.

Посылка ATAPI-устройству пакетной команды при помощи системного вызова ioctl выглядит следующим образом:

```
		ioctl(int fd, CDROM_SEND_PACKET, struct cdrom_generic_command *);
```

Первый параметр - дескриптор файла устройства. Второй параметр, CDROM_SEND_PACKET - спецификатор, указывающий на необходимость передачи устройству пакетной команды. Третий параметр - структура следующего типа:

```
/* for CDROM_PACKET_COMMAND ioctl */
struct cdrom_generic_command
{
	unsigned char		cmd[CDROM_PACKET_SIZE];
	unsigned char		*buffer;
	unsigned int		buflen;
	int			stat;
	struct request_sense	*sense;
	unsigned char		data_direction;
	int			quiet;
	int			timeout;
	void			*reserved[1];
};
```

Эта структура определена в файле <linux/cdrom.h>. Назначение полей структуры:
- cmd[CDROM_PACKET_SIZE] - 12-ти байтный командный пакет;
- buffer - указатель на буфер, куда будут помещены считанные данные. Также в этом буфере могут хранится данные, которые будут переданы устройству;
- buflen - размер передаваемых (принимаемых) данных;
- sense - структура, содержащая информацию о состоянии устройства (см. команду REQUEST SENSE и рис.17-18). Эта структура также определена в файле <linux/cdrom.h>;
- data_direction - направление обмена данными. Может принимать следующие значения:

```
#define CGC_DATA_WRITE 1 // передача данных устройству
#define CGC_DATA_READ 2 // прием данных от устройства
#define CGC_DATA_NONE 3 // нет обмена данными
```

- timeout - допустимое время выполнения команды.

Посылку пакетной команды ATAPI-устройству будет выполнять функция send_cmd. Параметрами функции являются:

- fd – дескриптор файла устройства
- cmd – указатель на 12-байтный командный пакет
- direction – направление обмена данными
- data – указатель на буфер для данных
- datalen – размер данных

```
int send_cmd(int fd, __u8 *cmd, unsigned char direction, __u8 *data, unsigned int datalen)
{
    struct cdrom_generic_command cgc;
    struct request_sense sense;

#define SK sense.sense_key
#define ASC sense.asc
#define ASCQ sense.ascq

    memset((void *)&cgc, 0, sizeof(struct cdrom_generic_command));
    memset(&sense, 0, sizeof(sense));

/* Заполняем поля структуры cgc */
    memcpy((void *)cgc.cmd, cmd, 12);
    cgc.sense = &sense;
    cgc.data_direction = direction;
    cgc.buffer = data;
    cgc.buflen = datalen;
    cgc.timeout = 20000;

/* Вызов системной функции IOCTL для передачи сформированной пакетной команды устройству */
    if(ioctl(fd, CDROM_SEND_PACKET, &cgc) < 0) {
      perror("ioctl CDROM_SEND_PACKET");

/* Если произошла ошибка, отобразим значения Sense Key, ASC и ASCQ для возможности анализа */
      printf("SK/ASC/ASCQ: 0x%02X/0x%02X/0x%02X\n", SK, ASC, ASCQ);
      return -1;
    }
    return 0;
}
```

Полный листинг функции send_cmd находится в файле SOURCE/ATAPI/LIB/send_cmd.c.

5.2 Открытие/закрытие лотка CD-привода

Приведем пример использования этой функции и рассмотрим программу открытия/закрытия лотка CD-привода.

```
#include <stdio.h>
#include <fcntl.h>
#include <linux/types.h>
#include <linux/cdrom.h>

#define CD_DRIVE "/dev/cdrom" // имя файла устройства
int fd; // дескриптор файла устройства

/* Функция проверки готовности устройства у принятию пакетной команды.
 * Для этого используется команда TEST UNIT READY, код 0x00
 */
void test_unit_ready()
{
    int i;
    __u8 test_unit_cmd[CDROM_PACKET_SIZE];

/* Формируем комадный пакет */
    memset(test_unit_cmd, 0, 12);
    test_unit_cmd[0] = GPCMD_TEST_UNIT_READY; // код команды, см. <linux/cdrom.h>

/* В течении 20 секунд опрашиваем устройство. При положительном результате возвращаемся
 * в вызывающую функцию, иначе выходим из программы
 */
    for(i = 0; i < 20; i++) {
```

```
        if(send_cmd(fd, test_unit_cmd, CGC_DATA_NONE, NULL, 0) == 0) return;
        sleep(1);
    }

    printf("Устройство не готово\n");
    exit(-1);
}

/* Функция открытия/закрытия лотка CD-привода. Параметр функции – значение бит LoEj/Start
 * командного пакета (см. рис.1)
 */
int start_stop_unit(unsigned char flag)
{
    __u8 cmd[CDROM_PACKET_SIZE];

/* Формируем командный пакет и посылаем устройству команду */
    memset((void *)cmd, 0, CDROM_PACKET_SIZE);
    cmd[0] = GPCMD_START_STOP_UNIT;
    cmd[4] = flag;
    if(send_cmd(fd, cmd, CGC_DATA_NONE, NULL, 0) < 0) return -1;
    return 0;
}

/* Главная функция */
int main()
{
/* Открываем файл устройства */
    fd = open(CD_DRIVE, O_RDONLY|O_NONBLOCK);
    if(fd < 0) { perror("open"); return -1; }

/* Команда открыть лоток */
    test_unit_ready();
    start_stop_unit(0x02);

    printf("Press ENTER");
    getchar();

/* Команда закрыть лоток */
    start_stop_unit(0x03);
    test_unit_ready();

    close(fd);
    return 0;
}
```

Полный текст программы открытия/закрытия лотка CD-привода находится в файле SOURCE/ATAPI/OPEN_CLOSE/open_close.c.

5.3 Чтение RAW TOC компакт-диска

Вернемся к рассмотрению команды READ TOC/PMA/ATIP. Нам понадобится компакт-диск, на котором записано несколько сессий. Наша задача - прочитать содержимое TOC этого диска, при условии, что поле Format содержит значение 10b. Формат данных TOC при Format Field = 10b, представлен на рис.21.

0	MSB	
1	TOC Data Length	LSB
2	First Complete Session Number	
3	Last Complete Session Number	
	TOC Track Descriptor(s)	
0	Session Number (Hex)	
1	ADR	Control
2	TNO	
3	POINT	
4	Min	
5	Sec	
6	Frame	

7	Zero
8	PMIN
9	PSEC
10	PFRAME

Рис.21. Формат данных TOC (Format Field = 10b)

Следующая структура описывает формат элемента записи TOC при Format = 10b:

```
struct toc {
       __u8 snum;          // номер сессии
       __u8 ctrl    :4;    // Control
       __u8 adr     :4;    // ADR
       __u8 tno;           // номер трека  (всегда 0)
       __u8 point;         // POINT
       __u8 min;           // AMIN
       __u8 sec;           // ASEC
       __u8 frame;         // AFRAME
       __u8 zero;          // 0
       __u8 pmin;          // PMIN
       __u8 psec;          // PSEC
       __u8 pframe;        // PFRAME
};
```

Сравните формат этой структуры с форматом Q-субканала Lead-In области, рис.14. Получается, что мы считываем «сырые» (RAW) данные Q-субканала Lead-In области. Считывания выполняет функция read_raw_toc:

```
/* Read TOC command */
int read_raw_toc()
{
    int i;
    __u8 cmd[12];
    __u8 *data_buff; // здесь будут сохранены результаты чтения TOC
    __u16 toc_data_len = 0; // размер записей TOC
    int toc_entries = 0; // число записей TOC
    int last; // номер последней сессии/трека
    __u32 lba;
    struct toc *t;

    memset(cmd, 0, 12);

/* Выделяем память для хранения данных TOC. Т.к. заранее размер данных
 * нам неизвестен, то зададим маскимальное значение - 64 kB
 */
    data_buff = (__u8 *)malloc(0xFFFF);
    memset(data_buff, 0, 0xFFFF);

/* Формируем пакетную команду */
    cmd[0] = GPCMD_READ_TOC_PMA_ATIP; // код команды (см. <linux/cdrom.h>)
    cmd[2] = 2; // поле Format Field = 10b
    cmd[7] = 0xFF; // размер считываемых
    cmd[8] = 0xFF; // данных

/* Посылаем устройству команду */
    if(send_cmd(fd, cmd, CGC_DATA_READ, data_buff, 0xFFFF) < 0) return -1;

/* Определяем размер данных TOC */
    *((__u8 *)&toc_data_len) = data_buff[1];
    *((__u8 *)&toc_data_len + 1) = data_buff[0];
    printf("TOC data length: %d\n", toc_data_len + 2);

/* Определяем число записей TOC */
    toc_entries = (toc_data_len - 2)/11;
    printf("TOC entries: %d\n", toc_entries);

/* Номер первой и последней завершенной сессии */
    printf("First: %d\t", data_buff[2]);
    last = data_buff[3];
```

```
    printf("Last: %d\n", last);

/* Выделяем память для данных TOC, размер этих данных уже точно известен */
    t = (struct toc *)malloc(toc_data_len + 2);
    memset((void *)t, 0, toc_data_len);

/* Копируем данные TOC из буфера data_buff
 * и освобождаем выделенную под него память
 */
    memcpy((void *)t, data_buff + 4, toc_data_len);
    free(data_buff);

#define SNUM(i) (t + i)->snum
#define ADR(i) (t + i)->adr
#define CTRL(i) (t + i)->ctrl
#define PMIN(i) (t + i)->pmin
#define PSEC(i) (t + i)->psec
#define PFRAME(i) (t + i)->pframe
#define MIN(i) (t + i)->min
#define SEC(i) (t + i)->sec
#define FRAME(i) (t + i)->frame
#define POINT(i) (t + i)->point

/* Отображаем результаты чтения TOC */
printf("Entry\tSession\tADR\tCTRL\tPoint\tMin\tSec\tFrame\tPMin\tPsec\tPFrame\tLBA\n");

    for(i = 0; i < toc_entries; i++) {
       printf("%d\t", i);
       printf("%d\t", SNUM(i));
       printf("%d\t", ADR(i));
       printf("%d\t", CTRL(i));
       printf("%X\t", POINT(i));
       printf("%d\t", MIN(i));
       printf("%d\t", SEC(i));
       printf("%d\t", FRAME(i));
       printf("%d\t", PMIN(i));
       printf("%d\t", PSEC(i));
       printf("%d\t", PFRAME(i));

       if((POINT(i) == 0xC0) || (POINT(i) == 0xC1)) {
           printf("\n");
           continue;
       }

/* Указатель B0 в полях Min/Sec/Frame содержит координаты начала следующей возможной
 * области программ. Переводим эти координаты в LBA-формат, с учетом того, что
 * перед треком находится Pre-gap область размером 150 секторов
 */
       if(POINT(i) == 0xB0) lba = MSF2LBA(MIN(i), SEC(i), FRAME(i)) + 150;
       else lba = MSF2LBA(PMIN(i), PSEC(i), PFRAME(i));

       printf("%u\n", lba);
    }

    free(t);
    return 0;
}
```

Полный текст программы чтения RAW TOC компакт-диска находится в файле SOURCE/ATAPI/TOC/read_raw_toc.c.

Устанавливаем в устройство компакт-диск, на котором создано 3 сессии, и запускаем на выполнение программу read_raw_toc. Вывод направим в файл toc:

```
       # ./read_raw_toc > toc
```

В результате в файле toc будут собраны данные Q-субканалов всех Lead-In областей компакт-диска:

```
TOC data length: 189
TOC entries: 17
First: 1     Last: 3
```

```
Число сессий на диске - 3
Entry   Session   ADR   CTRL   Point    Min     Sec    Frame   Pmin   Psec   PFrame     LBA
  0       1       1     4      A0       98      50     70      1      0      0        4350
  1       1       1     4      A1       98      47     14      1      0      0        4350
  2       1       1     4      A2       98      50     58      1      30     58       6658
  3       1       1     4      1        98      50     64      0      2      0         0
  4       1       5     4      B0       4       0      58      79     59     74       18058
  5       1       5     4      C0       102     0      156     97     32     10
  6       1       5     0      C1       72      0      0       0      0      0
  7       2       1     4      A0       3       27     31      2      0      0        8850
  8       2       1     4      A1       3       27     14      2      0      0        8850
  9       2       1     4      A2       3       25     19      4      27     47       19922
 10       2       1     4      2        3       27     25      4      2      58       18058
 11       2       5     4      B0       5       57     47      79     59     74       26822
 12       3       1     4      A0       4       58     20      3      0      0        13350
 13       3       1     4      A1       4       58     3       3      0      0        13350
 14       3       1     4      A2       4       58     8       24     27     15       109890
 15       3       1     4      3        4       58     14      5      59     47       26822
 16       3       5     4      B0       25      57     15      79     59     74       116790
```

Проведем анализ полученных результатов. Для этого воспользуемся таблицей 450 спецификации SCSI MMC-5 ([1]):

Table 450. TOC Track Descriptor Format, Q Sub-channel

CTRL	ADR	TNO	POINT	MIN	SEC	FRAME	ZERO	PMIN	PSEC	PFRAME
4 or 6	1	00h	01h-63h	ATIME (Absolute time)			00h	Start position of track		
4 or 6	1	00h	A0h	ATIME (Absolute time)			00h	First Track Number	Disc Type	00h
4 or 6	1	00h	A1h	ATIME (Absolute time)			00h	Last Track Number	00h	00h
4 or 6	1	00h	A2h	ATIME (Absolute time)			00h	Start position of Lead-out		
4 or 6	5	00h	B0h	Start time of next possible program in the Recordable Area of the disc			# of pointers in Mode 5	Maximum start time of outer-most Lead-out area in the Recordable Area of the disc		
4 or 6	5	00h	C0h	optimum recording power	Reserved	Reserved	Reserved	Start time of the first Lead-in Area of the disc		
4 or 6	5	00h	C1h	Copy of information from A1 point in ATIP.						

Таблица 6.

Запись	Point	Назначение
0	A0	запись содержит номер первого трека первой сессии. Номер трека находится в поле PMin и равен 1. В поле PSec содержится тип диска. Значение 0 соответствует режиму Mode 1
1	A1	запись содержит номер последнего трека первой сессии. Номер трека находится в поле PMin и равен 1, т.е. в сессии присутствует только один трек
2	A2	запись содержит стартовую позицию Lead-Out области сессии в полях PMin/PSec/PFrame
3	1	трек #1. Находится в первой сессии. Поля PMin/PSec/PFrame содержат координаты начала трека. Отметим, что координаты в формате LBA в TOC не хранятся, пересчет из MSF в LBA мы выполнили самостоятельно при помощи макроса MSF2LBA
4	B0	запись содержит координаты начала следующей возможной области программ в полях Min/Sec/Frame. Поля PMin/PSec/PFrame содержат максимальное возможное время Lead-Out области диска
5	C0	присутствует только в первой Lead-In области. Поле Min содержит значение оптимальной мощности записывающего лазера, поля PMin/PSec/PFrame - координаты начала первой Lead-In области диска
7	A0	запись содержит номер первого трека второй сессии. Номер трека находится в поле PMin и равен 2. Треки нумеруются последовательно. Так, если бы в первой сессии было два трека, то номер первого трека второй сессии был бы равен 3. Всего на диске может быть записано 99 треков. В поле PSec содержится тип диска (Mode 1)
8	A1	запись содержит номер последнего трека второй сессии. Номер трека находится в поле PMin и равен 2, т.е. во второй сессии, также как и в первой, присутствует только один трек
9	A2	запись содержит стартовую позицию Lead-Out области сессии в полях PMin/PSec/PFrame
10	2	трек #2. Находится во второй сессии. Поля PMin/PSec/PFrame содержат координаты начала трека

11	B0	координаты начала следующей возможной области программ в полях Min/Sec/Frame. Поля PMin/PSec/PFrame содержат максимальное возможное время Lead-Out области диска
12	A0	запись содержит номер первого трека третьей сессии. Номер трека находится в поле PMin и равен 3. В поле PSec содержится тип диска (Mode 1)
13	A1	запись содержит номер последнего трека третьей сессии. Номер трека находится в поле PMin и равен 3
14	A2	запись содержит стартовую позицию Lead-Out области сессии в полях PMin/PSec/PFrame
15	3	трек #3. Находится в третьей сессии. Поля PMin/PSec/PFrame содержат координаты начала трека
16	B0	координаты начала следующей возможной области программ в полях Min/Sec/Frame. Поля PMin/PSec/PFrame содержат максимальное возможное время Lead-Out области диска

5.4 Чтение сектора с компакт-диска

Повторим пример чтения сектора с компакт-диска. Этот пример аналогичен рассмотренному ранее, но с одним отличием – вместе с данными основного канала мы будем считывать данные Q-субканала кадра. Формат данных Q-субканала кадра области данных пользователя представлен на рис.22.

Байт	**Описание**
0	Control (4 MS bits), ADR (4 LS bits)
1	Номер трека
2	Индекс трека
3	Min
4	Sec
5	Frame
6	ZERO = 00h
7	AMin
8	ASec
9	AFrame
10-11	CRC or 00h
12-15	00h (pad)

Рис.22. Формат данных Q-субканала кадра области данных пользователя

Здесь Min, Sec, Frame - это координаты текущего кадра относительно начала трека, AMin, ASec, AFrame – координаты кадра относительно начала диска.

Чтение RAW-сектора выполняет функция read_cd. Входные параметры функции – номер сектора:

```
int read_cd(__u32 lba)
{
    int out_f;
    __u8 cmd[12];
    unsigned int datalen;
    __u32 sector;

#define QSCH_LEN 16 // размер данных Q-субканала

/* Буфер для данных. Значение CD_FRAMESIZE_RAW определено в файле <linux/cdrom.h>
 * и составляет 2352  - размер RAW-сектора
 */
    __u8 blk_buff[CD_FRAMESIZE_RAW + QSCH_LEN];

    memset(cmd, 0, 12);
    memset(blk_buff, 0, sizeof(blk_buff));

/* Формируем команду READ CD */
    cmd[0] = GPCMD_READ_CD; // код команды READ CD (см. <linux/cdrom.h>)
    cmd[1] = 0;  // читаем сектор любого типа
    cmd[8] = 1;  // число сеторов для чтения
    cmd[9] = 0xF8;  // из сетора считываем все поля
    cmd[10] = 2;  // считываем данные Q-субканала

    lba = __swab32(lba);
```

```
    memcpy((cmd + 2), (void *)&lba, 4);

/* Размер данных для чтения */
    datalen = CD_FRAMESIZE_RAW + QSCH_LEN;

/* Посылаем устройству команду */
    test_unit_ready();
    if(send_cmd(fd, cmd, CGC_DATA_READ, blk_buff, datalen) < 0) return -1;

/* Содержимое сектора сохраним в файле ./sector */
    out_f = open("sector", O_CREAT|O_RDWR|O_TRUNC, 0600);
    write(out_f, blk_buff, CD_FRAMESIZE_RAW);

/* Данные субканала сохраним в файле ./data_Q */
    out_f = open("data_Q", O_CREAT|O_RDWR|O_TRUNC, 0600);
    write(out_f, blk_buff + CD_FRAMESIZE_RAW, QSCH_LEN);

    close(fd); close(out_f);
    return 0;
}
```

Полный текст программы RAW-сектора находится в файле SOURCE/ATAPI/READ_CD/read_cd.c.

После запуска на выполнение программа сохранит данные основного канала в файле sector, данные Q-субканала - в файле data_Q. Посмотрим на результаты чтения, которые выдал привод TEAC CD-W524E для сектора номер 5555. Открываем в 16-ричном редакторе файл `sector` и смотрим на первые 16 байт:

```
00 FF FF FF FF FF FF FF FF FF FF 00 01 16 05 01
```

Байты 0–11 – это поле синхронизации Sync сектора. Следующие за ним 3 байта – это координаты сектора в формате MSF, значения представлены в BCD коде. Последний байт содержит значение режима записи данных, Data Mode.

Теперь посмотрим на результаты чтения данных Q-субканала, файл `data_Q`:

```
41 01 01 01 14 05 00 01 16 05 07 03 00 00 00 00
```

Байты 7, 8 и 9 содержат абсолютные координаты кадра в MSF формате, и значения координат совпадают со значениями, которые находятся в поле `Header` сектора.

5.5 Чтение субканальных данных

Для чтения данных субканала можно также использовать последовательность команд SEEK/READ SUBCHANNEL. Команда SEEK перемещает оптический элемент к нужному кадру, а READ SUBCHANNEL производит считывание необходимой нам информации. Считывать мы будем данные о текущей позиции оптического элемента, другими словами, координаты сектора, над которым элемент находится. Заодно мы посмотрим на точность позиционирования оптического элемента разных моделей CD-приводов.

Формат команды SEEK простой – байт 0 содержит код команды, байты 2-5 содержат координаты сектора в LBA формате, на который мы хотим позиционировать оптический элемент. Формат команды READ SUBCHANNEL представлен на рис.23.

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (42h)							
1	Reserved						MSF	Reserved
2	Reserved	SubQ	Reserved					
3	Sub-channel Data Format							
4	Reserved							
5	Reserved							
6	Track Number							
7	MSB							
8	Allocation Length							LSB
9-11	Reserved							

Рис.23. Формат команды READ SUBCHANNEL

Для чтения данных Q-субканала бит SubQ устанавливается в 1. Для чтения текущей позиции оптического элемента устройства поле Sub-channel Data Format должно содержать 01h. В результате выполнения команды устройство вернет блок данных следующего формата:

Bit / Byte	7	6	5	4	3	2	1	0
Sub Channel Data Header								
0	Reserved							
1	Audio Status							
2	MSB Sub-channel Data Length							
3	LSB							
CD-ROM Current Position Data Block								
4	Sub Channel Data Format Code (01h)							
5	ADR				Control			
6	Track Number							
7	Index Number							
8	MSB Absolute CD-ROM Address							
9								
10								
11	LSB							
12	MSB Track Relative CD-ROM Address							
13								
14								
15	LSB							

Рис.24. Формат данных о текущей позиции оптического элемента CD-привода

Заголовок блока содержит длину считанных из устройства данных. Координаты текущей позиции оптического элемента находятся в поле Absolute CD-ROM Address.

Формат данных о текущей позиции оптического элемента можно описать при помощи следующей структуры:

```
typedef struct {
   __u8 dfc;  // Data Format Code
   __u8 ctrl :4; // Control
   __u8 adr :4; // ADR
   __u8 tno;  // Track number
   __u8 ino;  // Index number
   __u8 a_addr[4]; // Absolute CD-ROM Address
   __u8 r_addr[4]; // Track Relative CD-ROM Address
} cur_pos_t;
```

Функция seek выполняет позиционирование оптического элемента на нужный сектор:

```
int seek(__u32 lba)
{
   __u8 cmd[12];

/* Формируем командный пакет */
   memset(cmd, 0, 12);
   cmd[0] = GPCMD_SEEK; // код команды SEEK
   lba = __swab32(lba);
   memcpy((void *)(cmd + 2), (void *)&lba, 4);

/* Посылаем устройству команду */
   test_unit_ready();
   if(send_cmd(fd, cmd, CGC_DATA_NONE, NULL, 0) < 0) return -1;

   return 0;
}
```

Чтение данных Q-субканала выполняет функция read_subch:

```
int read_subch()
{
   __u8 cmd[CDROM_PACKET_SIZE];
   __u8 data_buff[16]; // результаты чтения данных Q-субканала
   cur_pos_t *cur_pos;
   __u32 lba = 0;

   memset(data_buff, 0, 16);
   memset(cmd, 0, sizeof(cmd));

/* Формируем командный пакет */
   cmd[0] = GPCMD_READ_SUBCHANNEL; // код команды READ SUBCHANNEL
   cmd[1] = 2; // адреса заданы в MSF формате
   cmd[2] = 0x40; // бит SUBQ установлен - данные Q-субканала считываются
   cmd[3] = 1; // считывать информацию о текущей позиции опт.элемента
   cmd[8] = 16; // размер данных для чтения

/* Посылаем устройству команду */
   test_unit_ready();
   if(send_cmd(fd, cmd, CGC_DATA_READ, data_buff, 16) < 0) return -1;

   cur_pos = (void *)(data_buff + 4);

#define MIN cur_pos->a_addr[1]
#define SEC cur_pos->a_addr[2]
#define FRAME cur_pos->a_addr[3]

   printf("%d\t", cur_pos->ctrl);
   printf("%d\t", cur_pos->adr);
   printf("%X\t", cur_pos->tno);
   printf("%d\t", MIN);
   printf("%d\t", SEC);
   printf("%d\t", FRAME);

   lba = MSF2LBA(MIN, SEC, FRAME);
   printf("%u\n", lba);

   return;
}
```

Главная функция:

```
int main()
{
   int i;
   __u32 pos = 6655; // номер сектора, куда мы хотим позиционировать оптический элемент

   fd = open(CDROM, O_RDONLY|O_NONBLOCK);
   if(fd < 0) { perror("open"); return -1; }

   printf("NUM\tSECT\tCTRL\tADR\tTNO\tMIN\tSEC\tFRAME\tLBA\n");

   for(i = 0; i < 30; i++) {
       printf("%d\t%u\t", i, pos);
       seek(pos); // позиционируем оптический элемент
       sleep(2); // задержка, чтобы устройство успело подвести головку к заданной позиции
       read_subch(); // считываем данные субканала
       pos += 1; // следующий сектор
   }

   return 0;
}
```

Полный текст программы чтения данных Q-субканала находится в файле SOURCE/ATAPI/READ_Q_SCH/read_subch.c.

Рассмотрим пример работы программы. Устанавливаем в CD-привод компакт-диск, который мы ранее использовали для изучения формата данных Q-субканала Lead-In области (RAW TOC). Первая сессия на этом диске имеет размер 6658 блоков. Чтение данных Q-субканала выполнялось, начиная с сектора 6655 (см. значение переменной pos функции main). Вот какие результаты были получены при использовании привода TEAC CD-W524E:

ENTRY	SECT	CTRL	ADR	TNO	MIN	SEC	FRAME	LBA
0	6655	4	1	1	1	30	55	6655
1	6656	4	1	1	1	30	56	6656
2	6657	4	1	1	1	30	57	6657
3	6658	4	1	1	1	30	57	6657
4	**6659**	**4**	**1**	**AA**	**1**	**30**	**59**	**6659**
5	6660	4	1	AA	1	30	60	6660
6	6661	4	1	AA	1	30	61	6661
7	6662	4	1	AA	1	30	62	6662
8	6663	4	1	AA	1	30	63	6663

Начиная с сектора 6659 (запись 4) номер трека (поле TNO) меняется с 1 на 0xAA – это значит, что мы начали считывать данные Q-субканала Lead-Out области первой сессии.

Результаты чтения данных Q-субканала с использованием CD-привода ASUS DRW-1604P:

NUM	SECT	CTRL	ADR	TNO	MIN	SEC	FRAME	LBA
0	6655	4	1	1	1	30	50	6650
1	6656	4	1	1	1	30	56	6656
2	6657	4	1	1	1	30	56	6656
3	6658	4	1	1	1	30	54	6654
4	6659	4	1	AA	1	30	61	6661
5	6660	4	1	1	1	30	57	6657
6	6661	4	1	1	1	30	57	6657
7	6662	4	1	1	1	30	56	6656
8	6663	4	1	AA	1	30	63	6663
9	6664	4	1	AA	1	30	63	6663
10	6665	4	1	AA	1	30	61	6661
11	6666	4	1	AA	1	30	66	6666
12	6667	4	1	AA	1	30	67	6667
13	6668	4	1	AA	1	30	65	6665

Результаты работы программы показывают, что разные приводы выполняют команду позиционирования SEEK с разной точностью.

Кроме спецификатора CDROM_SEND_PACKET, в файле <linux/cdrom.h> определен целый набор специализированных команд, выполняющих определенное действие и не требующих формирования командного пакета в том объеме, который был нами рассмотрен в предыдущих примерах. Это в значительной степени упрощает работу с устройством. Например, команда для открытия и закрытия лотка CD-ROM выглядит следующим образом:

```
ioctl(fd, CDROMEJECT); // открыть лоток CD-ROM
ioctl(fd, CDROMCLOSETRAY); // закрыть лоток CD-ROM
```

Рассмотрим пример чтения таблицы содержания диска. Для этой цели в файле <linux/cdrom.h> определены два спецификатора - CDROMREADTOCHDR для чтения заголовка TOC и CDROMREADTOCENTRY для чтения записи TOC. Для хранения заголовка TOC в этом же файле (<linux/cdrom.h>) определена структура struct cdrom_tochdr следующего вида:

```
struct cdrom_tochdr
{
      __u8 cdth_trk0; /* start track */
      __u8 cdth_trk1; /* end track */
};
```

Для хранения записи TOC используется структура struct cdrom_tocentry:

```
struct cdrom_tocentry
{
      __u8 cdte_track;
      __u8 cdte_adr :4;
      __u8 cdte_ctrl :4;
      __u8 cdte_format;
```

```
        union cdrom_addr cdte_addr;
        __u8 cdte_datamode;
};
```

Объединение `union cdrom_addr` предназначено для хранения адресной информации сектора, и имеет следующий вид (см. <linux/cdrom.h>):

```
/* Address in either MSF or logical format */
union cdrom_addr
{
        struct cdrom_msf0 msf;
        int lba;
};
```

Структура `struct cdrom_msf0` содержит координаты сектора в MSF формате:

```
/* Address in MSF format */
struct cdrom_msf0
{
        __u8 minute;
        __u8 second;
        __u8 frame;
};
```

Вот что представляет из себя функция чтения TOC компакт-диска с использованием рассмотренных выше спецификаторов:

```
int main()
{
   int fd, i;

   struct cdrom_tochdr hdr; // заголовок содержимого TOC
   struct cdrom_tocentry toc; // дескриптор трека

/* Открываем файл устройства */
   fd = open(CD_DEVICE, O_RDONLY|O_NONBLOCK);
   if(fd < 0) { perror("open"); return -1; }

/* Считываем заголовок TOC */
   memset((void *)&hdr, 0, sizeof(struct cdrom_tochdr));
   if(ioctl(fd, CDROMREADTOCHDR, &hdr) < 0) {
       perror("ioctl CDROMREADTOCHDR");
       return -1;
   }

/* Поле cdth_trk0 структуры hdr содержит номер первого трека,
 * а поле cdth_trk1 - номер последнего трека. Отобразим эти значения
 */
   printf("First: %d\t", hdr.cdth_trk0);
   printf("Last: %d\n", hdr.cdth_trk1);

#define FIRST hdr.cdth_trk0
#define LAST hdr.cdth_trk1

/* Определим формат, в котором мы хотим получить координаты трека.
 * Для этого используется поле cdte_format структуры struct cdrom_tocentry
 */
   toc.cdte_format = CDROM_LBA;

/* Задавая в поле cdte_track структуры struct cdrom_tocentry
 * последовательно номера треков от первого до последнего, мы
 * определяем их стартовые координаты в формате LBA
 */
   for(i = FIRST; i <= LAST; i++) {
       toc.cdte_track = i;
       if(ioctl(fd, CDROMREADTOCENTRY, &toc) < 0) {
              perror("ioctl CDROMREADTOCENTRY");
              return -1;
       }
```

```
        printf("track: %d\t", i);
        printf("lba: %d\n", toc.cdte_addr.lba);
    }

    return 0;
}
```

Полный текст программы чтения таблицы содержания диска находится в файле SOURCE/ATAPI/IOCTL2/get_cd_toc.c.

Для чтения субканальных данных кадра используется спецификатор *CDROMSUBCHNL*. Данные субканала помещаются в структуру struct cdrom_subchnl следующего вида:

```
struct cdrom_subchnl
{
        __u8 cdsc_format;
        __u8 cdsc_audiostatus;
        __u8 cdsc_adr :4;
        __u8 cdsc_ctrl :4;
        __u8 cdsc_trk;
        __u8 cdsc_ind;
        union cdrom_addr cdsc_absaddr;
        union cdrom_addr cdsc_reladdr;
};
```

Следующая программа каждые 2 секунды отслеживает текущее положение оптического элемента, отображает координату текущего сектора в форматах LBA и MSF, а также номер трека.

```
#define CD_DEVICE "/dev/cdrom"
int main()
{
    int fd;
    struct cdrom_subchnl sc;

/* Открываем файл устройства */
    fd = open(CD_DEVICE, O_RDONLY|O_NONBLOCK);
    if(fd < 0) { perror("open"); return -1; }

* Проверяем тип компакт-диска. Это должен быть Audio-CD */
    if(ioctl(fd, CDROM_DISC_STATUS) != CDS_AUDIO) {
        printf("I need Audio_CD!\n");
        return 0;
    }

/* Считывание координат производим в бесконечном цикле */
    for(;;) {
        sc.cdsc_format = CDROM_LBA; // формат адреса - LBA
        ioctl(fd, CDROMSUBCHNL, &sc); // считываем данные Q-субканала

/* Отображаем номер текущего трека и координату кадра в формате LBA */
       printf("Track: %d\t", sc.cdsc_trk);
       printf("LBA: %d\t", sc.cdsc_absaddr.lba);

/* То же самое - для формата MSF */
        sc.cdsc_format = CDROM_MSF;
        ioctl(fd, CDROMSUBCHNL, &sc);

       printf("MSF: %d %d %d\n", sc.cdsc_absaddr.msf.minute, \
                    sc.cdsc_absaddr.msf.second, \
                    sc.cdsc_absaddr.msf.frame);

/* Ждем две секунды и повторяем. Выход по Ctrl-C */
        sleep(2);
    }
    return 0;
}
```

Полный текст программы находится в файле SOURCE/ATAPI/IOCTL2/read_sch.c.

Теперь делаем вот что – устанавливаем в привод Audio-CD, в одной консоли запускаем проигрыватель аудио компакт-дисков (workbone, например), а в другой - программу read_sch. Через каждые 10-15 секунд воспроизведения переходим к следующему треку на компакте, и смотрим на результаты работы read_sch. Получается примерно следующая картина:

```
Track: 1        LBA: 178        MSF: 0 4 28
Track: 1        LBA: 329        MSF: 0 6 29
Track: 1        LBA: 480        MSF: 0 8 30
Track: 1        LBA: 630        MSF: 0 10 31
Track: 1        LBA: 781        MSF: 0 12 31
Track: 1        LBA: 932        MSF: 0 14 32
Track: 2        LBA: 14748      MSF: 3 18 48
Track: 2        LBA: 14898      MSF: 3 20 48
Track: 3        LBA: 15049      MSF: 3 22 49
Track: 4        LBA: 36509      MSF: 8 8 59
Track: 4        LBA: 36659      MSF: 8 10 60
Track: 4        LBA: 36810      MSF: 8 12 60
Track: 4        LBA: 36961      MSF: 8 14 61
```

Результаты работы программы красноречиво свидетельствуют о постоянном изменении текущей координаты.

В качестве примера чтения секторов с компакт-диска рассмотрим программу считывания музыкальных треков с Audio-CD. Для этой цели используется спецификатор CDROMREADAUDIO и структура struct cdrom_read_audio:

```
struct cdrom_read_audio
{
       union cdrom_addr addr; // frame address
       __u8 addr_format; // CDROM_LBA or CDROM_MSF
       int nframes;  // number of 2352-byte-frames to read at once
       __u8 *buf;  // frame buffer (size: nframes*2352 bytes)
};
```

Считанный трек преобразуется в WAV-формат. Для этого необходимо сформировать заголовок WAV-файла. Формат заголовка описывает структура вида (взята из исходных текстов программы cdda2wav):

```
typedef struct {
   __u8 riff[4];
   __u32 size;
   __u8 wave[8];
   __u32 size1;
   __u16 format_tag;
   __u16 channels;
   __u32 sample_per_sec;
   __u32 byte_per_sec;
   __u16 block_align;
   __u16 bit_per_sample;
   __u8 data[4];
   __u32 size2;
} wav_header_t;
```

Программа чтения аудиотреков выглядит следующим образом:

```
int main()
{
   int fd, out, n;
   unsigned int i, start_lba, end_lba;
   wav_header_t w_hdr;

   __u8 buff[CD_FRAMESIZE_RAW];

   struct stat s;
   struct cdrom_tochdr hdr;
   struct cdrom_tocentry toc;
   struct cdrom_read_audio cda;

   fd = open(CD_DEVICE, O_RDONLY|O_NONBLOCK);
   if(fd < 0) { perror("open"); return -1; }
```

/ Проверяем тип компакт-диска. Это должен быть Audio-CD */*

```
   if(ioctl(fd, CDROM_DISC_STATUS) != CDS_AUDIO) {
       printf("I need Audio_CD!\n");
       return 0;
   }

/* Определяем число треков на компакт-диске */
   memset((void *)&hdr, 0, sizeof(struct cdrom_tochdr));
   if(ioctl(fd, CDROMREADTOCHDR, &hdr) < 0) {
       perror("ioctl CDROMREADTOCHDR");
       return(errno);
   }

   printf("First: %d\t", hdr.cdth_trk0);
   printf("Last: %d\n", hdr.cdth_trk1);

#define FIRST hdr.cdth_trk0
#define LAST hdr.cdth_trk1

/* Вводим номер трека, который мы хотим считать с диска */
   printf("Enter track number: ");
   scanf("%d", &n);

   if((n < 1) || (n > LAST)) {
       printf("Wrong track number\n");
       return -1;
   }

/* Задаем формат адреса LBA и считываем стартовые координаты трека */
   toc.cdte_format = CDROM_LBA;
   toc.cdte_track = n;

   if(ioctl(fd, CDROMREADTOCENTRY, &toc) < 0) {
       perror("ioctl CDROMREADTOCENTRY");
       return -1;
   }

   start_lba = toc.cdte_addr.lba; // стартовый адрес трека

/* Конечный адрес трека определим как стартовый адрес следующего трека. Если мы выбрали
 * последний трек на диске, то необходимо определить начало Lead-Out области диска
 */
   if(n == LAST) toc.cdte_track = CDROM_LEADOUT;
   else toc.cdte_track = n + 1;

   if(ioctl(fd, CDROMREADTOCENTRY, &toc) < 0) {
       perror("ioctl CDROMREADTOCENTRY");
       return -1;
   }

   end_lba = toc.cdte_addr.lba; // конечный адрес трека

/* Создаем файл track.wav для хранения считанных аудиоданных */
   out = open("./track.wav", O_CREAT|O_RDWR|O_TRUNC, 0600);

/* В начале файла должен находится заголовок установленного формата
 * длиной 44 байта. Но так как нам пока неизвестны все значения полей
 * заголовка (в частности, размер файла), запишем в файл пустой заголовок
 */
   memset(&w_hdr, 0, sizeof(wav_header_t));
   write(out, (void *)&w_hdr, WAV_HDR_LEN);

/* Начинаем считывать аудиоданные. При каждом обращении к диску считываем
 * один кадр (2352 байта), адресация в формате LBA, считанные данные помещаем
 * в буфер buff, а затем записываем в файл track.wav
 */
   cda.addr_format = CDROM_LBA;
   cda.nframes = 1;
   cda.buf = buff;

   printf("Track size: %d sectors\n", end_lba - start_lba);
```

```
    for(i = start_lba; i < end_lba; i++) {

        memset(buff, 0, sizeof(buff));
        cda.addr.lba = i;
        fprintf(stdout,"\rLBA: %u", i - start_lba + 1);
        fflush(stdout);

        /* Читаем аудиоданные*/
        if(ioctl(fd, CDROMREADAUDIO, &cda) < 0) {
               perror("ioctl CDROMREADAUDIO");
               return -1;
        }

        write(out, (__u8 *)buff, CD_FRAMESIZE_RAW);
    }

    printf("\n");

/* Определяем размер файла track.wav */
    memset(&s, 0, sizeof(struct stat));
    if(fstat(out, &s) < 0) { perror("fstat"); exit(-1); }

/* Формируем WAV-заголовок и записываем его в начало файла */
    memset(&w_hdr, 0, sizeof(wav_header_t));
    memcpy(w_hdr.riff, "RIFF", 4);
    w_hdr.size = s.st_size - 8;
    memcpy(w_hdr.wave, "WAVEfmt ", 8); // последний символ - пробел
    w_hdr.size1 = 16;
    w_hdr.format_tag = 1;
    w_hdr.channels = 2;
    w_hdr.sample_per_sec = 44100;
    w_hdr.byte_per_sec = 176400;
    w_hdr.block_align = 4;
    w_hdr.bit_per_sample = 16;
    memcpy(w_hdr.data, "data", 4);
    w_hdr.size2 = s.st_size - WAV_HDR_LEN;

    lseek(out, 0, 0);
    write(out, (void *)&w_hdr, WAV_HDR_LEN);

    printf("OK\n");

    close(fd); close(out);
    return 0;
}
```

Полный исходный текст программы чтения аудиотреков находится в файле SOURCE/ATAPI/IOCTL2/cd2wav.c.

6. Доступ к ATAPI-устройству при помощи SCSI Generic драйвера

6.1 Общие сведения о SCSI Generic драйвере

SCSI Generic драйвер (далее sg-драйвер) входит в состав ядра ОС Linux, и позволяет приложению пользователя посылать пакетные команды устройству, при условии, что устройство эти команды понимает. Доступ к sg-драйверу выполняется через специальные файлы устройства, которые находятся в каталоге /dev. Список некоторых файлов можно получить при помощи команды:

```
# ls -l /dev/sg[01]
crw-------  1 root  root  21, 0 Apr 13 /dev/sg0
crw-------  1 root  root  21, 1 Apr 13 /dev/sg1
```

Каждый файл соответствует одному подключенному SCSI-устройству.

Обращение к SCSI-устройству через sg-драйвер выполняется при помощи системного вызова ioctl следующим образом:

```
ioctl(sg_fd, SG_IO, struct sg_io_hdr *)
```

Первый параметр sg_fd - это дескриптор файла sg-устройства /dev/sg*, открытого при помощи системного вызова open( ). Третий параметр - структура следующего типа:

```
typedef struct sg_io_hdr
{
     int interface_id; /*'S' for SCSI generic (required) */
     int dxfer_direction; /* data transfer direction  */
     unsigned char cmd_len; /* SCSI command length ( <= 16 bytes) */
     unsigned char mx_sb_len; /* max length to write to sbp */
     unsigned short iovec_count; /* 0 implies no scatter gather */
     unsigned int dxfer_len; /* byte count of data transfer */
     void *dxferp; /* points to data transfer memory or scatter gather list */
     unsigned char *cmdp; /* points to command to perform */
     unsigned char *sbp; /* points to sense_buffer memory */
     unsigned int timeout; /* MAX_UINT->no timeout (unit: millisec) */
     unsigned int flags; /* 0 -> default, see SG_FLAG... */
     int pack_id; /* unused internally (normally) */
     void *usr_ptr; /* unused internally */
     unsigned char status; /* scsi status */
     unsigned char masked_status;/* shifted, masked scsi status */
     unsigned char msg_status; /* messaging level data (optional) */
     unsigned char sb_len_wr; /* byte count actually written to sbp */
     unsigned short host_status; /* errors from host adapter */
     unsigned short driver_status; /* errors from software driver */
     int resid; /* dxfer_len - actual_transferred */
     unsigned int duration; /* time taken by cmd (unit: millisec) */
     unsigned int info; /* auxiliary information */
} sg_io_hdr_t; /* 64 bytes long (on i386) */
```

Данная структура определена в файле <scsi/sg.h>. Назначение основных полей структуры:
- interface_id - это поле должно содержать литеру 'S';
- dxfer_direction - направление передачи данных. Поле может принимать следующие значения (см. <scsi/sg.h>):

#define SG_DXFER_NONE (-1) - нет обмена данными
#define SG_DXFER_TO_DEV (-2) - передача данных устройству
#define SG_DXFER_FROM_DEV (-3) - прием данных от устройства

- cmdp - указатель на командный пакет, посылаемый устройству;
- cmd_len - размер командного пакета. Если для ATAPI-устройств размер командного пакета фиксирован и равен 12 байт, то для SCSI-устройств размер пакета может принимать значения 6, 10, 12 и 16 байт;
- sbp - указатель на буфер SENSE DATA (данные о состоянии устройства после выполнения команды, [1,5]);
- mx_sb_len - максимальный размер буфера SENSE DATA;
- sb_len_wr - реальный размер данных, сохраненных в буфере SENSE DATA;
- dxferp - указатель на буфер для данных, принимаемых от устройства или передаваемых устройству
- dxfer_len - размер передаваемых/принимаемых данных
- iovec_count - если это поле равно 0, то буфер для данных представляет собой простой линейный массив байт, и поле dxferp – указатель на этот массив. В противном случае iovec_count содержит число элементов в массиве, на который указывает поле dxferp. Элементом этого массива является структура следующего вида:

```
typedef struct sg_iovec {
     void * iov_base; /* starting address */
     size_t iov_len;  /* length in bytes */
} sg_iovec_t,
```

Подробная информация о SCSI Generic драйвере приведена в SCSI-Generic-HOWTO [4].

6.2 Пример использования SCSI Generic драйвера - чтение PMA диска

Рассмотрим порядок использования sg-драйвера на примере чтения PMA (Program Memory Area) CD-R/RW диска. Этот пример очень похож на пример чтения таблицы содержания диска TOC. Для чтения PMA используется таже самая команда - READ TOC/PMA/ATIP. Формат этой команды был рассмотрен на рис.19. Поле Format командного пакета должно содержать значение 0011b. В ответ на команду READ TOC/PMA/ATIP устройство вернет блок данных следующего формата:

READ TOC/PMA/ATIP response data (Format = 0011b)

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)			PMA Data Length				
1								(LSB)
2-3	Reserved							
PMA Descriptor(s)								
0	Reserved							
1	ADR				CONTROL			
2	TNO							
3	POINT							
4	Min							
5	Sec							
6	Frame							
7	Zero							
8	PMIN							
9	PSEC							
10	PFRAME							

Рис.25. Формат данных PMA, Format Field = 0011b

Этот блок данных можно описать при помощи следующей структуры:

```
typedef struct {
    __u8 rez;         // reserved
    __u8 ctrl :4;     // Control
    __u8 adr :4;      // ADR
    __u8 tno;         // TNO (always 0)
    __u8 point;       // POINT
    __u8 min;         // AMIN
    __u8 sec;         // ASEC
    __u8 frame;       // AFRAME
    __u8 zero;        // 0
    __u8 pmin;        // PMIN
    __u8 psec;        // PSEC
    __u8 pframe;      // PFRAME
} __attribute__ ((packed)) pma_t;
```

Поле PMA Data Length содержит размер данных PMA, при этом длина самого поля (2 байта) не учитывается. Назначение каждого байта дескриптора PMA определяется значением поля ADR:

Таблица 7.

ADR	Поле	Значение поля
1	TNO	0
	POINT	номер трека в BCD-коде
	MIN,SEC,FRAME	координаты конца трека
	PMIN,PSEC,PFRAME	координаты начала трека
2	TNO	0
	POINT	0
	MIN,SEC,FRAME	идентификатор диска (6 цифр в BCD-коде)
	PMIN	0
	PSEC	формат сессии: 00 - CDDA, CD-ROM, 10 - CD-I, 20 - CD-ROM-XA
	PFRAME	0

Обмен данными с sg-драйвером выполняет функция send_cmd. Параметры функции:

- sg_fd – дескриптор открытого файла устройства;
- cmd - указатель на командный пакет;
- cmdlen - длина командного пакета;
- direction - направление передачи данных;
- data - указатель на блок памяти для данных, передаваемых устройству или принимаемых от устройства. Если обмен данными не предусмотрен, этот параметр содержит значение NULL;
- datalen - размер блока данных, на который указывает data. Если data == NULL, то datalen == 0

- timeout – допустимое время выполнения команды

```
int send_cmd(int sg_fd, __u8 *cmd, __u8 cmdlen, int direction, __u8 *data, \
    __u32 datalen, unsigned int timeout)
{
   sg_io_hdr_t io_hdr;

/* Буфер для хранения информации о состоянии устройства после выполнения команды */
   __u8 sense_buffer[32];

#define SK (sense_buffer[2] & 0x0F)
#define ASC sense_buffer[12]
#define ASCQ sense_buffer[13]

/* Формируем запрос к sg-драйверу - заполняем поля структуры sg_io_hdr_t
 * необходимыми значениями
 */
   memset(&io_hdr, 0, sizeof(sg_io_hdr_t));
   io_hdr.interface_id = 'S';
   io_hdr.cmd_len = cmdlen; // длина команды
   io_hdr.mx_sb_len = sizeof(sense_buffer);
   io_hdr.dxfer_direction = direction; // направление передачи данных
   io_hdr.dxfer_len = datalen; // размер данных
   io_hdr.dxferp = data; // указатель на блок данных
   io_hdr.cmdp = cmd;// указатель на командный пакет
   io_hdr.sbp = sense_buffer;
   io_hdr.timeout = timeout * 1000; /* 20000 millisecs == 20 seconds */

/* Посылаем устройству команду */
   if(ioctl(sg_fd, SG_IO, &io_hdr) < 0) {
       perror("SG_IO ioctl"); return -1;
   }

/* Если при выполнении команды произошла ошибка, то сохраним
 * значения SK/ASC/ACSQ для возможности установления причины ошибки
 */
   if((io_hdr.info & SG_INFO_OK_MASK) != SG_INFO_OK) {
       if(io_hdr.sb_len_wr > 0) {
             syslog(LOG_INFO, "Command: 0x%02X", io_hdr.cmdp[0]);
             syslog(LOG_INFO, "Sense data (SK/ASC/ASCQ):\
                         0x%02X/0x%02X/0x%02X", SK, ASC, ASCQ);
       }
       return -1;
   }
   return 0;
}
```

Исходный текст функции send_cmd находится в файле SOURCE/SG/LIB/send_cmd.c.

Чтение PMA-области диска выполняет функция read_pma:

```
int read_pma()
{
   int i, k;
   __u8 read_pma_cmd[10];
   __u8 *pma_data_buff; // здесь будут сохранены результаты чтения PMA
   __u16 pma_data_len = 0;  // реальная длина записей PMA
   int pma_entries = 0; // число записей PMA
   __u32 lba;

   pma_t *pma;

   pma_data_buff = (__u8 *)malloc(0xFFFF);
   memset(pma_data_buff, 0, 0xFFFF);

/* Формируем командный пакет */
   memset(read_pma_cmd, 0, 10);
   read_pma_cmd[0] = 0x43;  // код команды READ_TOC/PMA/ATIP
   read_pma_cmd[2] = 3;  // Format Field = 011b, считываем PMA
```

```
   read_pma_cmd[7] = 0xFF;
   read_pma_cmd[8] = 0xFF;

/* Проверяем готовность устройства к приему команды и посылаем ему командный пакет */
   test_unit_ready(sg_fd);
   if(send_cmd(sg_fd, read_pma_cmd, 10, SG_DXFER_FROM_DEV,\
                   pma_data_buff, 0xFFFF, 200) < 0) return -1;

/* Считываем размер записей PMA */
   *((__u8 *)&pma_data_len) = pma_data_buff[1];
   *((__u8 *)&pma_data_len + 1) = pma_data_buff[0];
   printf("PMA data length: %d\n", pma_data_len + 2);

/* Определяем число записей PMA */
   pma_entries = (pma_data_len - 2)/11;
   printf("PMA entries: %d\n", pma_entries);

/* Размер данных PMA точно известен и равен pma_data_length. Выделяем
 * блок память для данных PMA и копируем их туда из pma_data_buff.
 * После этого можно освободить блок памяти, на который указывает pma_data_buff
 */
   pma = (pma_t *)malloc(pma_data_len);
   memset((void *)pma, 0, pma_data_len);
   memcpy((void *)pma, pma_data_buff + 4, pma_data_len);
   free(pma_data_buff);

/* Отображаем данные PMA */
   printf("Entry\tADR\tCTRL\tPoint\tZero\tMin\tSec\tFrame\tPMin\tPsec\tPFrame\tLBA\n");

#define PMIN(i) (pma + i)->pmin
#define PSEC(i) (pma + i)->psec
#define PFRAME(i) (pma + i)->pframe
#define ADR(i) (pma + i)->adr

   for(i = 0; i < pma_entries; i++) {
       printf("%d\t", i);
       printf("%X\t", ADR(i));
       printf("%X\t", (pma + i)->ctrl);
       printf("%X\t", (pma + i)->point);
       printf("%d\t", (pma + i)->zero);
       printf("%d\t", (pma + i)->min);
       printf("%d\t", (pma + i)->sec);
       printf("%d\t", (pma + i)->frame);
       printf("%d\t", PMIN(i));
       printf("%d\t", PSEC(i));
       printf("%d\t", PFRAME(i));

       lba = MSF2LBA(PMIN(i), PSEC(i), PFRAME(i));

       if(ADR(i) != 1) printf("---\n");
       else printf("%u\n", lba);
   }
   free(pma);
   return 0;
}
```

Вызов функции read_pma для чтения данных PMA области диска выполняется из главной функции:

```
int main()
{
/* Открываем файл устройства. Обратите внимание на режим открытия – чтение/запись.
 * Если устройство открыто в режиме «Только чтение» (O_RDONLY), то оно воспринимает только команды
 * INQUIRY, TEST UNIT READY, REQUEST SENSE, READ CAPACITY, READ BUFFER, READ(6) (10) and (12),
 * MODE SENSE(6) and (10)
 */
    if((sg_fd = open(SG_DEV, O_RDWR)) < 0) {
        perror("open");
        return -1;
    }
```

```
/* Считываем PMA */
    if(read_pma() < 0) printf("Cannot read PMA\n");

    close(sg_fd);
    return 0;
}
```

Текст данной программы находится в файле SOURCE/SG/READ_PMA/read_pma.c.

Устанавливаем в устройство диск CD-RW, на котором записано 3 аудиотрека, и запускаем программу на выполнение. Получаем следующий результаты:

```
PMA data length - 46
PMA entries - 4
Entry ADR     CTRL   Point  Zero   Min    Sec    Frame  PMin   Psec   PFrame LBA
0     2       0      0      0      54     88     82     0      0      0             ---
1     1       0      1      0      5      3      16     0      2      0             0
2     1       0      2      0      8      28     58     5      5      16            22741
3     1       0      3      0      12     30     21     8      30     58
```

Поля MIN/SEC/FRAME записи 0 содержат идентификатор диска в BCD-коде. Поле PSEC содержит формат сессии. В нашем примере это значение равно 0 - CD-DA, аудиодиск. Записи 1-3 в поле POINT содержат номер трека, поля PMIN/PSEC/PFRAME содержат координаты начала трека, MIN/SEC/FRAME - координаты конца трека. Хорошо видно, что пауза между треками составляет 2 секунды.

7. Свойства, профили и страницы режимов устройства

7.1 Свойства и профили устройства

Прежде чем послать устройству какую-либо команду, надо убедиться в том, что устройство способно эту команду выполнить. Для этого необходимо установить, какие именно команды поддерживает устройство. Набор команд, поддерживаемых устройством, называется свойством (Features).

Одно устройство может поддерживать несколько свойств. Базовый набор свойств устройства называется профилем (Profile). Перечень всех существующих свойств и профилей приведен в спецификации SCSI MMC-5, п.5 "Features and Profiles for Multi-Media Device".

Для того чтобы выяснить, обладает ли устройство тем или иным свойством, ему необходимо послать команду GET CONFIGURATION. Данная команда определена в спецификациях SCSI MMC-5 и INF-8090i ([1,2]), и позволяет получить полный список свойств, поддерживаемых устройством, и текущий профиль устройства. Текущий профиль определяет, какие именно свойства доступны на данный момент. По команде GET CONFIGURATION устройство вернет блок данных, состоящий из заголовка свойства (Feature Header) и списка дескрипторов свойств (Feature Descriptors):

GET CONFIGURATION response data format

Bit / Byte	7	6	5	4	3	2	1	0
0-7	Feature Header							
8-n	Feature Descriptor(s)							

Рис.26. Формат блока данных, возвращаемого по команде GET CONFIGURATION

Формат заголовка свойства Feature Header представлен на рис.27:

Feature Header

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
1	Data Length							
2								
3								(LSB)
4-5	Reserved							
6	(MSB)			Current Profile				
7								(LSB)

Рис.27. Формат заголовка свойства

Поле Data Length содержит размер считанных данных, следующих за эти полем, в поле Current Profile находится значение текущего профиля устройства. Если свойство не поддерживается устройством, то команда GET CONFIGURATION вернет только заголовок свойства, и поле Data Length будет содержать значение 4 (2 поля Reserved по одному байту каждое + 2 байта поля Current Profile).

Общий формат дескриптора свойства Feature Descriptor представлен на рис.28:

Feature Descriptor generic format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB) Feature Code							
1								(LSB)
2	Reserved			Version			Persistent	Current
3	Additional Length							
4-n	Feature Dependent Data							

Рис.28. Общий формат дескриптора свойства

Назначение полей дескриптора свойства:
- Feature Code - код свойства. Каждое свойство имеет свой уникальный код. Список всех кодов приведен в спецификации SCSI MMC-5, п.5.3 "Feature Definitions"
- Persistent - если этот бит установлен в 0, то данное свойство может менять текущий статус. Если бит равен 1, то свойство всегда активно
- Current - если бит установлен в 1, то данное свойство активно, т.е. устройство поддерживает набор команд, определенный этим свойством
- Feature Dependent Data - данные, специфичные для указанного свойства

Формат команды GET CONFIGURATION приведен на рис.29.

GET CONFIGURATION CDB

Bit / Byte	7	6	5	4	3	2	1	0
0	OPERATION CODE (46h)							
1	Reserved							RT
2	(MSB) Starting Feature Number							
3								(LSB)
4-6	Reserved							
7	(MSB) Allocation Length							
8								(LSB)
9	Control							

Рис.29. Формат команды GET CONFIGURATION

Назначение полей командного пакета:
- RT - определяет тип данных, возвращаемых устройством. Поле может принимать следующие значения:
  - 00b - устройство возвращает заголовок свойства и дескрипторы всех свойств, поддерживаемых устройством, даже если свойство не является активным
  - 01b - устройство возвращает заголовок свойства и дескрипторы активных свойств устройства (у которых бит Current установлен в единицу)
  - 10b - устройство возвращает заголовок и дескриптор свойства, номер которого задан в поле Starting Feature Number. Если запрашиваемое свойство не поддерживается устройством, возвращается только заголовок свойства Feature Header (рис.27)
  - 11b - зарезервировано

- Allocation Length - размер памяти, выделенной для данных

Рассмотрим пример. Необходимо выяснить, может ли устройство выполнить запись треков на компакт-диск в режиме TAO. Для этого надо установить, обладает ли устройство свойством "CD Track at Once". Код этого свойства равен 0x002D (см. [1]). Дескриптор свойства "CD Track at Once" представлен на рис.30.

CD Track at Once Feature Descriptor Format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB) Feature Code = 002Dh							
1								(LSB)
2	Reserved			Version = 2h			Persistent	Current
3	Additional Length = 04h							
4	Resvd	BUF	Resvd	R-W Raw	R-W Pack	Test Write	CD-RW	R-W Sub-code
5	Reserved							
6	(MSB) Data Type Supported							
7								(LSB)

Рис.30. Формат дескриптора свойства "CD Track at Once"

Формат дескриптора свойства "CD Track at Once" описывает следующая структура:

```
typedef struct {
   __u16 f_code;
   __u8 current :1;
   __u8 persistent :1;
   __u8 version :4;
   __u8 res1 :2;
   __u8 add_length;
   __u8 rw_subcode :1;
   __u8 cd_rw :1;
   __u8 test_write :1;
   __u8 rw_pack :1;
   __u8 rw_raw :1;
   __u8 res2 :1;
   __u8 BUF :1;
   __u8 res3 :1;
   __u8 reserved;
   __u16 data_type_support;
} __attribute__ ((packed)) cd_tao_t;
```

Устройство способно выполнить запись трека на компакт-диск в режиме TAO в том случае, если бит CD-RW установлен в единицу.

Рассмотрим функцию, выполняющую проверку наличия у устройства свойства "CD Track at Once". Функция принимает один параметр - код проверяемого свойства :

```
int check_feature(__u16 f_num)
{
   __u8 get_conf_cmd[10];
   __u8 data_buff[16]; // результаты чтения
   __u32 data_len = 0; // реальная длина данных
   __u16 cur_prof = 0; // текущий профиль, Current Profile

   cd_tao_t *cd_tao;

   memset(data_buff, 0, sizeof(data_buff));
   cd_tao = (void *)(data_buff + 8);

/* Формируем командный пакет */
   memset(get_conf_cmd, 0, 10);
   get_conf_cmd[0] = 0x46; // код команды GET CONFIGURATION
   get_conf_cmd[1] = 2; // RT = 10b
   get_conf_cmd[2] = *((__u8 *)&f_num + 1);
   get_conf_cmd[3] = *((__u8 *)&f_num);
   get_conf_cmd[8] = 16;

/* Посылаем устройству командный пакет */
   test_unit_ready(sg_fd);
   if(send_cmd(sg_fd, get_conf_cmd, 10, SG_DXFER_FROM_DEV,\
                   data_buff, 16, 200) < 0) return -1;

   memcpy((void *)&data_len, data_buff, 4);
```

```
    data_len = __swab32(data_len);
    if(data_len == 4) return -1; // свойство не поддерживается

/* Текущий профиль */
    *((__u8 *)&cur_prof) = data_buff[7];
    *((__u8 *)&cur_prof + 1) = data_buff[6];
    printf("\nCurrent profile: 0x%04X\n", cur_prof);

/* Значение бита CD-RW */
    printf("CD-RW bit: %d\n", cd_tao->cd_rw);

/* Значение бита Current */
    printf("Current bit: %d\n", cd_tao->current);

    return 0;
}
```

Полный текст программы проверки наличия у устройства свойства "CD Track at Once" находится в файле SOURCE/SG/GET_CONF/check_feature.c.

Установив в привод CD-RW диск и запустив программу на выполнение, получаем следующий результат:

```
Current profile: 0x000A
CD-RW bit: 1
Current bit: 1
```

Биты CD-RW и Current установлены. Это значит, что проверяемое свойство поддерживается устройством и активно в настоящий момент. Значение текущего профиля устройства – CD-RW, см. п. 5.4.10 «Profile 000Ah: CD-RW» спецификации SCSI MMC-5 ([1]). Список свойств, соответствующих данному профилю, приведен в этом же пункте, в таблице 185 "Mandatory Feature for CD-RW". Свойство "CD Track at Once" входит в их число.

Заменим диск CD-RW на CD-ROM и снова запустим программу на выполнение. Результаты работы программы:

```
Current profile: 0x0008
CD-RW bit: 1
Current bit: 0
```

Свойство "CD Track at Once" устройством поддерживается, но текущий профиль – CD-ROM, см. п. 5.4.8 «Profile 0008h: CD-ROM» - не позволяет это свойство применить, поэтому бит Current сброшен. Список свойств, соответствующих профилю "CD-ROM", приведен в таблице 181 "Mandatory Feature for CD-ROM", п. 5.4.8 спецификации SCSI MMC-5 ([1]).

7.2 Страницы режимов

Для установки или определения параметров работы устройства используются страницы режимов (Mode Page). Каждая страница режима характеризуется кодовым номером, длиной и набором параметров. В таблице 8 приведен список кодов некоторых страниц режимов. Полный перечень находится в спецификации SCSI MMC-5, п.7.1.3 "Mode Pages".

Таблица 8

Код страницы	Назначение страницы
0x01	параметры режима коррекции ошибок чтения/записи
0x05	параметры режима записи
0x0D	параметры CD-дисковода
0x0E	параметры управления звуковоспроизведением
0x1A	параметры энергопотребления
0x1C	управление сообщениями о сбоях и неисправностях
0x1D	time-out и защита

Прием и передача страниц осуществляется при помощи списка параметров режима Mode Parameter List. Список страниц начинается с заголовка (Mode Parameter Header), за которым могут следовать одна или несколько страниц режимов (рис.31).

Mode Parameter List

Bit / Byte	7	6	5	4	3	2	1	0

0-7	Mode Parameter Header
8-n	Mode Page(s)

Рис.31. Формат списка Mode Parameter List

Формат заголовка списка страниц приведен на рис.32.

Mode Parameters Header

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB) Mode Data Length							
1								(LSB)
2-5	Reserved							
6	(MSB) Block Descriptor Length = 0							
7								(LSB)

Рис.32. Формат заголовка списка страниц (Mode Parameter Header)

Поле Mode Data Length содержит размер блока данных, следующего сразу за этим полем, т.е. 6 оставшихся байт заголовка плюс длину запрашиваемой страницы. Если страница передается устройству, то значение поля Mode Data Length не используется (зарезервировано).

7.3 Страница параметров режима записи

Рассмотрим подробнее, что из себя представляет страница режима, на примере страницы параметров режима записи, Write Parameters Mode Page. Формат страницы параметров режима записи представлен на рис.33:

Write Parameters Page

Bit / Byte	7	6	5	4	3	2	1	0
0	PS	Reserved	Page Code (05h)					
1	Page Length (32h or 36h)							
2	Reserved	BUFE	LS_V	Test Write	Write Type			
3	Multi-session		FP	Copy	Track Mode			
4	Reserved				Data Block Type			
5	Link Size							
6	Reserved							
7	Reserved		Initiator Application Code					
8	Session Format							
9	Reserved							
10	(MSB) Packet Size							
11								
12								
13								(LSB)
14	(MSB) Audio Pause Length							
15								(LSB)
16	(MSB) ... Media Catalog Number ...							
17								
……								
30								
31								(LSB)
32	(MSB) ... International Standard Recording Code ...							
33								
……								
46								
47								(LSB)
48	Sub-header Byte 0							
49	Sub-header Byte 1							
50	Sub-header Byte 2							
51	Sub-header Byte 3							
52-55	Vendor Specific							

Рис.33. Формат страницы параметров режима записи

Назначение полей:
- Page Code - код страницы, содержит значение 0x05

- Page Length - размер страницы в байтах
- Write Type - используемый режим записи:
    01 - Track-at-once (TAO)
    02 - Session-at-once (SAO)
    03 - RAW

- Test Write - режим имитации записи данных на носитель. Бит имеет значение только для режимов TAO и SAO
- Track Mode - содержит значение поля управления Control Field Q-субканала при ADR = 1 (см. "Организация данных на компакт-диске"). Мы будем использовать только два значения этого поля:
    0 – аудио-трек
    4 – трек данных

- Multi-session - показывает, как закрытие текущей сессии влияет на открытие следующей:
    00b - указатель B0 отсутствует в TOC, открытие новой сессии запрещено
    01b - указатель B0 содержит значение FF:FF:FF, открытие новой сессии запрещено
    11b - открытие новой сессии разрешено. Указатель B0 содержит координаты начала следующей области программ

- Data Block Type - определяет тип блока данных и его размер. Перечень всех значений этого поля находится в таблице 610 "Data Block Type Codes" спецификации SCSI MMC. Мы будем использовать только два значения:
    0 - блок "сырых" данных размером 2352 байт (блок аудиоданных)
    8 - блок данных размером 2048 байт, формат блока данных Mode 1

- Session Format - формат сессии. Это значение записывается в поле PSEC указателя A0 таблицы содержания диска TOC. Поле может принимать следующие значения:
    00h - диск CD-DA или CD-ROM
    01h - диск CD-I
    20h - диск CD-ROM XA

В поле Sub-Header устройству передается информация для заполнения подзаголовка сектора при записи данных на диск в формате Mode 2 Form 1(2).

7.4 Определение и установка параметров устройства

Для определения текущих параметров устройства применяется команда MODE SENSE, а для установки параметров - команда MODE SELECT. Формат команды MODE SENSE приведен на рис.34.

Mode Sense CDB

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (5Ah)							
1	Reserved			LLBAA	DBD	Reserved		
2	PC		Page Code					
3-6	Reserved							
7	(MSB)	Allocation Length						
8								(LSB)
9	Control							

Рис.34. Формат команды MOSE SENSE

Поле Page Code содержит код запрашиваемой страницы режимов, поле Allocation Length - размер считываемых данных.

Формат команды MODE SELECT приведен на рис.35.

Mode Select CDB

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (55h)							
1	Reserved			PF=1	Reserved			SP
2-6	Reserved							
7	(MSB)	Parameter List Length						
8								(LSB)
9	Control							

Рис.35. Формат команды MODE SELECT

Поле Parameter List Length содержит размер передаваемого списка параметров в байтах. Бит PF (Page Format) установлен в единицу. Это означает, что формат страницы соответствует стандарту SCSI.

Рассмотрим две функции, mode_sense и mode_select, при помощи которых мы сможем управлять параметрами устройства в режиме записи. Определим структуру для описания формата страницы параметров режима записи:

```
typedef struct {
   __u8 page_code   :6;
   __u8 rez         :1;
   __u8 ps          :1;
   __u8 page_length;
   __u8 write_type  :4;
   __u8 test_write  :1;
   __u8 ls_v        :1;
   __u8 BUFE        :1;
   __u8 rez1        :1;
   __u8 track_mode  :4;
   __u8 copy        :1;
   __u8 FP          :1;
   __u8 multises    :2;
   __u8 dbt         :4;
   __u8 rez2        :4;
   __u8 link_size;
   __u8 rez3;
   __u8 hac         :6;
   __u8 rez4        :2;
   __u8 s_format;
   __u8 rez5;
   __u32 packet_size;
   __u16 apl;
   __u8 mcn[16];
   __u8 isrc[16];
   __u32 sh;
} __attribute__ ((packed)) wpm_t;
```

```
/*  Определим глобальные переменные */
__u16 page5_len; // размер списка параметров режима
__u8 *page5_data; // данные списка параметров режима
wpm_t *wpm; // данные страницы параметров записи
```

Функция mode_sense выполняет чтение страницы параметров режима записи. Считанные данные сохраняются в структуре wpm.

```
int mode_sense()
{
   __u8 mode_sense_cmd[10];

/* Определяем размер списка параметров режима. Для этого считываем первые два байта
 * заголовка списка страниц – поле Mode Data Length (рис.32).
 */
   page5_data = (__u8 *)malloc(2);
   memset(mode_sense_cmd, 0, 10);
   mode_sense_cmd[0] = MODE_SENSE_10;  // код команды MODE_SENSE
   mode_sense_cmd[2] = 5; // страница параметров записи
   mode_sense_cmd[8] = 2; // размер поля Mode Data Length

/* Ждем готовность устройства и отправляем ему команду */
   test_unit_ready(sg_fd);
   if(send_cmd(sg_fd, mode_sense_cmd, 10, SG_DXFER_FROM_DEV, \
                   page5_data, 2, 200) < 0) return -1;

   *((__u8 *)&page5_len) = page5_data[1];
   *((__u8 *)&page5_len + 1) = page5_data[0];
   page5_len += 2; // учитываем размер самого поля Mode Data Length

/* Выделяем память для списка параметров */
   free(page5_data);
```

```
    page5_data = (__u8 *)malloc(page5_len);

    memset(page5_data, 0, page5_len);
    wpm = (void *)(page5_data + 8); // wpm указывает на начало страницы параметров режима записи

/* Формируем команду MODE SENSE */
    memset(mode_sense_cmd, 0, 10);
    mode_sense_cmd[0] = MODE_SENSE_10; // код команды MODE_SENSE
    mode_sense_cmd[2] = 5; // номер страницы параметров записи
    mode_sense_cmd[8] = page5_len; // размер считываемых данных

/* Ждем готовность устройства и отправляем ему команду */
    test_unit_ready(sg_fd);
    if(send_cmd(sg_fd, mode_sense_cmd, 10, SG_DXFER_FROM_DEV, \
                    page5_data, page5_len, 200) < 0) return -1;

    return 0;
}
```

Установку требуемых параметров режима записи выполняет функция mode_select. Параметры этой функции: тип блока данных (поле dbt структуры wpm_t) и тип информации, находящейся в треке (поле track_mode структуры wpm_t):

```
int mode_select(__u8 dbt, __u8 track_mode)
{
    __u8 mode_select_cmd[10];

/* Считываем страницу параметров режима записи */
    if(mode_sense() < 0) exit(-1);

/* Устанавливаем требуемые параметры режима записи */
    wpm->write_type = 1; // режим записи – TAO
    wpm->dbt = dbt;
    wpm->s_format = 0;
    wpm->track_mode = track_mode;

/* Формируем команду MODE SELECT */
    memset(mode_select_cmd, 0, 10);
    mode_select_cmd[0] = MODE_SELECT_10; // код команды Mode Select
    mode_select_cmd[1] = 0x10;
    mode_select_cmd[7] = *((__u8 *)&page5_len + 1);
    mode_select_cmd[8] = *(__u8 *)&page5_len;

    test_unit_ready(sg_fd);
    if(send_cmd(sg_fd, mode_select_cmd, 10, SG_DXFER_TO_DEV, \
                    page5_data, page5_len, 200) < 0) return -1;

    free(page5_data);
    return 0;
}
```

8. Запись данных на CD-R/RW диск.

8.1 Запись односессионного диска с данными в режиме TAO

Рассмотрим, как выполнить запись данных на компакт-диск. Под термином "данные" понимается файл в формате ISO (файл-образ), полученный при помощи утилиты mkisofs, или каким-либо другим способом.

Общий алгоритм выполнения операции записи следующий:

- устанавливаются необходимые параметры режима записи – тип и размер блока данных, режим трека (аудио/данные) и др.
- резервируется пространство для трека при помощи команды RESERVE TRACK. Размер трека равен размеру файла-образа в блоках.
- из файла-образа считываются блоки данных, и команда WRITE_10 выполняет запись этих блоков в соответствующие сектора компакт-диска. Для односессионного диска стартовый номер сектора, с которого начинается запись, равен 0.
- по окончании записи данных (достигнут конец файла-образа) последовательно выполняются команды SYNCHRONIZE CACHE, CLOSE TRACK, CLOSE SESSION. По команде SYNCHRONIZE CACHE все данные,

подлежащие записи, переносятся на носитель, внутренний буфер устройства освобождается. Команда CLOSE TRACK закрывает трек, а CLOSE SESSION завершает сессию - формирует Lead-In и Lead-Out области сессии (диска).

Формат RESERVE TRACK команды представлен на рис.36.

RESERVE TRACK CDB

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (53h)							
1-4	Reserved							
5	(MSB)							
6				Reservation				
7				Size				
8								(LSB)
9	Control Byte							

Рис.36. Формат команды RESERVE TRACK

В поле Reservation size указывается размер трека в блоках. Каждый вызов команды RESERVE TRACK модифицирует PMA-область, добавляя в нее запись о координатах нового трека.

Формат WRITE_10 команды представлен на рис.37.

WRITE (10) CDB

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (2Ah)							
1	Reserved			DPO	FUA	Reserved		RelAdr
2	(MSB)							
3				Logical Block Address				
4								
5								(LSB)
6	Reserved							
7	(MSB)			Transfer Length				
8								(LSB)
9	Control							

Рис.37. Формат команды WRITE_10

Назначение полей:
- Logical Block Address – стартовый адрес блока, с которого будет выполнятся запись информации
- Transfer Length - число блоков для записи.

В команде SYNCHRONIZE CACHE используется только нулевой байт - он содержит код операции, значение 0x35.

Формат команды CLOSE TRACK/SESSION представлен на рис.38.

CLOSE TRACK/SESSION CDB

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (5Bh)							
1	Reserved							IMMED
2	Reserved					Close Function		
3	Reserved							
4	(MSB)			Track Number				
5								(LSB)
6-8	Reserved							
9	Control Byte							

Рис.38. Формат команды CLOSE TRACK/SESSION

Поле Close Function может принимать следующие значения:
- 001b - закрыть трек, номер которого указан в поле Track Number
- 010b - закрыть последнюю незавершенную сессию, значение поля Track Number игнорируется

Реализуем алгоритм записи данных на компакт-диск при помощи функции do_record. Входные параметры функции – имя файла-образа:

```
void do_record(__u8 *image_name)
{
   struct stat s;
   __u32 track_size;

/* Определяем размер файла в блоках по 2048 байт */
   memset((void *)&s, 0, sizeof(struct stat));
   if(stat(image_name, &s)) { perror("stat"); return; }
   track_size = s.st_size/CD_FRAMESIZE;

   printf("Размер файла %s: %u Кбайт", image_name, s.st_size/1024);
   printf(" (%u блоков)\n", track_size);

/* Устанавливаем параметы режима записи: записываем данные, размер блока - 2048 байт (Data Mode 1) */
   if(mode_select(8, 4) < 0) return;

/* Резервируем пространство для трека */
   if(reserv_track(track_size) < 0) {
       printf("\nОшибка резервирования трека\n");
       return;
   }

/* Записываем файл-образ на диск */
   if(write_iso(image_name) < 0) {
       printf("Ошибка записи файла-образа %s\n", image_name);
       return;
   }

/* Закрываем трек */
   if(close_track() < 0) {
       printf("Ошибка закрытия трека\n");
       return;
   }

/* Закрываем сессию */
   if(close_session() < 0) {
       printf("Ошибка закрытия сессии\n");
       return;
   }
   printf("OK\n");

/* Извлекаем диск из лотка */
   eject_cd();

   return;
}
```

Резервирование пространства для трека выполняет функция reserv_track. Входные параметры функции – размер трека в блоках.

```
int reserv_track(__u32 track_size)
{
   __u8 reserv_track_cmd[10];
   __u32 size = 0;

   memset(reserv_track_cmd, 0, 10);
   reserv_track_cmd[0] = 0x53; // код команды RESERVE TRACK

/* Заполняем поле Reservation size */
   size = __swab32(track_size);
   memcpy((void *)(reserv_track_cmd + 5), (void *)&size, 4);

/* Посылаем устройству команду */
   test_unit_ready(sg_fd);
   if(send_cmd(sg_fd, reserv_track_cmd, 10, SG_DXFER_NONE, NULL, 0, 200) < 0) return -1;
```

```
   return 0;
}
```

Запись данных на диск выполняет функция write_iso:

```
#define SECT_NUM 10 // число секторов для записи
#define BUFF_SIZE (CD_FRAMESIZE * SECT_NUM)

int write_iso(__u8 *file_name)
{
   int in_f;
   __u8 write_cmd[10];
   __u8 write_buff[BUFF_SIZE];
   __u32 lba = 0, lba1 = 0;
   ssize_t bsize = 0;

/* Открываем файл-образ */
   in_f = open(file_name, O_RDONLY);

/* Формируем команду WRITE_10 */
   memset(write_buff, 0, BUFF_SIZE);
   memset(write_cmd, 0, 10);
   write_cmd[0] = WRITE_10;
   write_cmd[8] = SECT_NUM;

   test_unit_ready(sg_fd);

   for(;;) {

/* Считываем из файла-образа блоки данных */
       bsize = read(in_f, write_buff, BUFF_SIZE);
       if(bsize == 0) {
              fprintf(stdout,"\rLBA: %6d\n", lba1); fflush(stdout);
              return 0;
       }

       if(bsize < BUFF_SIZE) break;
       fprintf(stdout,"\rLBA: %6d", lba1); fflush(stdout);

/* Заполняем поле Logical Block Address */
       lba = __swab32(lba1);
       memcpy((write_cmd + 2), (void *)&lba, 4);
       lba1 += SECT_NUM;

/* Посылаем устройству команду */
       if(send_cmd(sg_fd, write_cmd, 10, SG_DXFER_TO_DEV, write_buff, \
                     BUFF_SIZE, 200) < 0) return -1;
   }

   write_cmd[8] = bsize / CD_FRAMESIZE;

   fprintf(stdout,"\rLBA: %6d", lba1); fflush(stdout);
   lba = __swab32(lba1);
   memcpy((write_cmd + 2), (void *)&lba, 4);

   if(send_cmd(sg_fd, write_cmd, 10, SG_DXFER_TO_DEV, write_buff, bsize, 200) < 0) return -1;

   fprintf(stdout,"\rLBA: %6d\n", lba1 + write_cmd[8]);
   fflush(stdout);

   return 0;
}
```

По окончании записи данных необходимо послать устройству три команды: SYNCHRONIZE CACHE, CLOSE TRACK, CLOSE SESSION. Команду SYNCHRONIZE CACHE формирует и посылает устройству функция sync_cache():

```
int sync_cache()
{
   __u8 flush_cache[10];
```

```
   memset(flush_cache, 0, 10);
   flush_cache[0] = SYNCHRONIZE_CACHE; // код команды

   if(send_cmd(sg_fd, flush_cache, 10, SG_DXFER_NONE, NULL, 0, 200) < 0) return -1;

   return 0;
}
```

Закрывает трек функция close_track:

```
int close_track()
{
   __u8 close_trk_cmd[10];

   if(sync_cache() < 0) printf("\n-ERR: SYNCHRONIZE CACHE\n");

/* Формируем команду закрытия трека */
   memset(close_trk_cmd, 0, 10);
   close_trk_cmd[0] = 0x5B; // код команды
   close_trk_cmd[2] = 1; // флаг закрытия трека
   close_trk_cmd[5] = 1; // номер трека

/* Ждем готовность устройства и посылаем ему команду */
   test_unit_ready(sg_fd);
   if(send_cmd(sg_fd, close_trk_cmd, 10, SG_DXFER_NONE, NULL, 0, 200) < 0) return -1;

   return 0;
}
```

Функция close_session закрывает текущую сессию – формирует Lead-In и Lead-Out области:

```
int close_session()
{
   __u8 close_sess_cmd[10];

   memset(close_sess_cmd, 0, 10);
   close_sess_cmd[0] = 0x5B;
   close_sess_cmd[2] = 2; // флаг закрытия текущей сессии

   test_unit_ready(sg_fd);
   if(send_cmd(sg_fd, close_sess_cmd, 10, SG_DXFER_NONE, NULL, 0, 200) < 0) return -1;

   return 0;
}
```

8.2 Обработка ошибочных ситуаций

При использовании привода TEAC CD-W524E программа, рассмотренная выше, выполняет запись данных на компакт-диск без ошибок. Попытка записи информации с помощью провода ASUS DRW-1604P заканчивается неудачей - программа аварийно завершает выполнение со следующим сообщением:

```
Sense data (SK/ASC/ASCQ): 0x02/0x04/0x08
Driver status=0x28
```

Здесь Sense Key = 0x02, ASC = 0x04, ASCQ = 0x08, что означает LOGICAL UNIT NOT READY, LONG WRITE IN PROGRESS (устройство не готово к работе, выполняется операция длинной записи). Остановка выполнения программы всегда происходит на секторе номер 320. Аналогичным образом ведут себя приводы MITSUMI и _NEC. С приводом TEAC, как уже было сказано, подобной проблемы не возникало. В чем причина этой ошибки? Попытаемся разобраться.

Каждый привод имеет внутренний буфер для данных, и при записи информация сначала попадает в этот буфер, а затем из него переносится на диск. Буфер имеет ограниченную ёмкость, определить которую можно при помощи команды READ BUFFER CAPACITY. Формат этой команды приведён на рис.39.

READ BUFFER CAPACITY CDB

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation Code (5Ch)							

1	Reserved	BLOCK
2-6	Reserved	
7	(MSB) Allocation Length	
8		(LSB)
9	Control	

Рис.39. Формат команды READ BUFFER CAPACITY

Если поле BLOCK установлено в 0, устройство вернёт блок данных следующего формата:

Buffer Capacity Structure, when Block = 0

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB) Data Length							
1								(LSB)
2-3	Reserved							
4	(MSB) Length of the Buffer							
5								
6								
7								(LSB)
8	(MSB) Available Length of Buffer							
9								
10								
11								(LSB)

Рис.40. Формат блока данных о буфере устройства, BLOCK = 0

Здесь Length of the Buffer - размер буфера в байтах, Available Length of Buffer - размер неиспользуемой (доступной) области буфера в байтах.

Модифицируем листинг из второй части - при записи информации будем контролировать размер доступной области внутреннего буфера устройства при помощи функции read_buff_cap():

```
void read_buff_cap()
{
      __u8 read_buff_cap_cmd[10];
      __u8 data_buff[12];
      __u32 buff_alen = 0; // размер доступной области буфера

      memset(read_buff_cap_cmd, 0, 10);
      read_buff_cap_cmd[0] = 0x5C;
      read_buff_cap_cmd[8] = 0x0C;

      send_cmd(read_buff_cap_cmd, 10, SG_DXFER_FROM_DEV, data_buff, 0x0C, 20);

      memcpy((void *)&buff_alen, data_buff + 8, 4);
      buff_alen = __swab32(buff_alen);

      printf("\tAvailable length - %u\n", buff_alen);
      return;
}
```

Контроль за размером доступной области буфера будет выполняться в функции write_iso, в цикле записи информации (полный листинг не приводится):

```
int write_iso(__u8 *file_name)
{
      ....

      while(read(in_f, write_buff, CD_FRAMESIZE) > 0) {

            read_buff_cap();

            printf("lba - %6d", lba1);
            lba = __swab32(lba1);
            memcpy((write_cmd + 2), (void *)&lba, 4);
            lba1 += 1;
```

```
            send_cmd(write_cmd, 10, SG_DXFER_TO_DEV, write_buff, CD_FRAMESIZE, 20);

      }
      return 0;
}
```

Результат работы программы (привод ASUS DRW-1604P):

```
      Available length - 1267712
lba -  0      Available length - 1175552
lba -  1      Available length - 1173504
lba -  2      Available length - 1171456
........
lba -  317    Available length - 526336
lba -  318    Available length - 524288
lba -  319    Available length - 0
lba - 320
Sense data: 0x70 0x00 0x02 0x00 0x00 0x00 0x00 0x0e 0x00 0x00
 0x00 0x00 0x04 0x08 0x00 0x00 0x00 0x00 0x00 0x00
 0x00 0x00
Driver status=0x28
Cannot write image.iso
```

Как только доступная область буфера становиться равной нулю, устройство на попытку передачи ему информации отвечает LOGICAL UNIT NOT READY, LONG WRITE IN PROGRESS. Один из вариантов решения данной проблемы - подождать, пока буфер устройства освободится, а затем продолжить запись, например:

```
      while(read_buff_cap() == 0) continue;
```

Такой вариант устраивает ASUS, но MITSUMI и _NEC так просто не сдаются, и, попав в цикл, не хотят его покидать. Спецификация SFF8090i ([2]) при возникновении такого рода ошибки требует повторить команду записи (см. п.14.48 WRITE(10) command):

While writing is occurring, if WRITE (10) command or WRITE (12) command cannot be terminated immediately due to insufficient buffer capacity, the logical unit may terminate the WRITE command with CHECK CONDITION status, 2/04/08 LOGICAL UNIT NOT READY, LONG WRITE IN PROGRESS and **the host shall issue the same WRITE command again.** After logical unit becomes ready due to sufficient buffer capacity for the WRITE command, the WRITE command shall be performed normally.

Именно таким образом поступает утилита cdrecord. Запустив ее с ключем -v, мы увидим следующую картину:

```
Executing 'write_g1' command on Bus 0 Target 1,
      Lun 0 timeout 200s
CDB: 2A 00 00 00 3A 01 00 00 1F 00

cdrecord: Input/Output error, write_g1: scsi sendcmd: no error
CDB: 2A 00 00 00 3A 01 00 00 1F 00

Status: 0x2 (CHECK CONDITION)
Sense buffer: 70 00 02 00 00 00 00 0E 00 00 00 00
      04 08 00 00
Sense Key: 0x2 Not Ready, Segment 0
Sense Code: 0x04 Qual 0x08 (logical unit not ready,
      Long write in progress) Fru 0x0
Sense flags: BLK 0 (not valid)
cmd finished after 0x000 s timeout 200s

Executing 'write_g1' command on Bus 0 Target 1,
      Lun 0 timeout 200s
CDB: 2A 00 00 00 3A 01 00 00 1F 00
cmd finished after 0x000 s timeout 200s
```

CDB - это Command Descriptor Block (дескриптор командного блока), 2A - код команды WRITE_10. Зафиксировав ошибку LONG WRITE IN PROGRESS, программа cdrecord в соответствии с требованием спецификации повторно посылает устройству команду записи.

Исходя из вышеизложенного, внесём исправления в текст функции посылки пакетной команды устройству send_cmd:

```
int send_cmd(int sg_fd, __u8 *cmd, __u8 cmdlen, int direction, __u8 *data, \
    __u32 datalen, unsigned int timeout)
{
   sg_io_hdr_t io_hdr;
   __u8 sense_buffer[32];

#define SK (sense_buffer[2] & 0x0F)
#define ASC sense_buffer[12]
#define ASCQ sense_buffer[13]

   memset(&io_hdr, 0, sizeof(sg_io_hdr_t));
   io_hdr.interface_id = 'S';
   io_hdr.cmd_len = cmdlen;
   io_hdr.mx_sb_len = sizeof(sense_buffer);
   io_hdr.dxfer_direction = direction;
   io_hdr.dxfer_len = datalen; // размер данных
   io_hdr.dxferp = data; // указатель на блок данных
   io_hdr.cmdp = cmd;
   io_hdr.sbp = sense_buffer;
   io_hdr.timeout = timeout * 1000;

rep:
   if(ioctl(sg_fd, SG_IO, &io_hdr) < 0) {
      perror("SG_IO ioctl");
      return -1;
   }

   if((io_hdr.info & SG_INFO_OK_MASK) != SG_INFO_OK) {
       if(io_hdr.sb_len_wr > 0) {

/* Если SK/ASC/ASCQ == 02/04/08 (Not Ready. Long Write in Progress),
 * то необходимо повторить команду WRITE_10. Для этого выполняется переход на метку rep
 */
             if((SK == NOT_READY) && (ASC == 0x04) && (ASCQ == 0x08)) goto rep;
             syslog(LOG_INFO, "Command: 0x%02X", io_hdr.cmdp[0]);
             syslog(LOG_INFO, "Sense data (SK/ASC/ASCQ): \
                         0x%02X/0x%02X/0x%02X", SK, ASC, ASCQ);
       }
      return -1;
   }
   return 0;
}
```

Полный тескт программы для записи односессионного диска с данными находится в файле SOURCE/SG/WRITE/TAO/DATA/iso2cd_tao.c.

Вторая ошибка касается SCSI Generic драйвера. Дело в том, что неправильно сформированный запрос к драйверу может вывести операционную систему из строя. Так, если в команде не требующей обмена данными с устройством (например TEST UNIT READY), указатель на буфер для данных dxferp (см. 6.1) не будет равен NULL, то система рухнет:

```
      __u8 test_unit_cmd[6];
      __u8 buff[10];

      memset(test_unit_ready, 0, 6);
      send_cmd(sg_fd, test_unit_cmd, 6, SG_DXFER_NONE, buff, 0, 200)
```

В вызове `send_cmd` указатель **`buff`** не равен NULL, и запуск этого кода приведет к краху системы.

Для устранения этой ошибки я внес маленькое дополнение в код SG-драйвера, в файл linux/drivers/scsi/sg.c. Находим в этом файле функцию sg_new_write, вызов которой выглядит следующим образом:

```
static ssize_t sg_new_write(Sg_fd * sfp, const char * buf, size_t count,
      int blocking, int read_only, Sg_request ** o_srp)
{
   int                   k;
   Sg_request          * srp;
   sg_io_hdr_t         * hp;
   unsigned char         cmnd[sizeof(dummy_cmdp->sr_cmnd)];
   int                   timeout;
```

В эту функцию, после вызова

```
hp = &srp->header;
__copy_from_user(hp, buf, SZ_SG_IO_HDR);
```

добавляем проверку

```
if((hp->dxfer_direction == SG_DXFER_NONE) && (hp->dxferp != NULL)) return -1;
```

после чего пересобираем ядро.

На этом закончим изучение односессионных CD-дисков и рассмотрим, что нужно сделать для того, чтобы создать многосессионный диск.

8.3 Запись многосесссионного диска

Для создания многосессионного диска необходимо присвоить полю Multi-session страницы параметров режима записи значение 11b. В этом случае при формировании Lead-In области указатель B0 таблицы содержания диска (Table of Contents, TOC) будет содержать координаты начала следующей области программ (формат TOC и пример чтения были рассмотрены выше). Формирование образа, который будет записан в первым, ничем не отличается от формирования образа для односессионного диска. Каждый новый записываемый образ должен содержать ссылку на предыдущий, чтобы драйвер файловой системы мог собрать содержимое всех сессий в единое целое. Для этого при формировании ISO-образа (кроме первого) в опциях утилиты mkisofs необходимо указать стартовые координаты первого трека последней сессии и координаты следующей возможной области программ. Назовем эти координаты A и B. При помощи cdrecord значения A и B определяются следующим образом:

```
cdrecord -dev=X,Y,Z -msinfo
```

где X, Y и Z - это координаты устройства, msinfo – сокращение от multi-session information. После выполнения команды мы получим два числа - искомые стартовые координаты. Эти числа необходимо указать в соответствующих опциях mkisofs (опция -C, см. man mkisofs) при подготовки образа для записи.

Рассмотрим два способа определения значений чисел A и B. Оба способа подразумевают чтение таблицы содержания диска (TOC), только в первом случае мы читаем "сырую" таблицу (RAW TOC), в во втором - информацию о сессиях.

При помощи функции read_raw_toc прочитаем RAW TOC:

```
/* Формат записи TOC */
typedef struct {
      __u8 snum;            // Session Number
      __u8 ctrl      :4;    // Control
      __u8 adr       :4;    // ADR
      __u8 tno;             // TNO (always 0)
      __u8 point;           // POINT
      __u8 min;             // AMIN
      __u8 sec;             // ASEC
      __u8 frame;           // AFRAME
      __u8 zero;            // 0
      __u8 pmin;            // PMIN
      __u8 psec;            // PSEC
      __u8 pframe;          // PFRAME
} toc_t;

int read_raw_toc()
{
      int i;
      __u8 read_toc_cmd[10];
      __u8 *data_buff; // результаты чтения TOC
      __u16 toc_data_len = 0; // длина записей TOC
      __u32 lba;
      int toc_entries = 0; // число записей TOC
      int last = 0; // номер последней сессии
      toc_t *t;

      data_buff = (__u8 *)malloc(0xFFFF);
```

```
      memset(data_buff, 0, 0xFFFF);
      memset(read_toc_cmd, 0, 10);

      read_toc_cmd[0] = READ_TOC;
      read_toc_cmd[2] = 2; // Format Field = 10b – читаем RAW TOC
      read_toc_cmd[7] = 0xFF;
      read_toc_cmd[8] = 0xFF;

      send_cmd(sg_fd, read_toc_cmd, 10, SG_DXFER_FROM_DEV, data_buff, 0xFFFF, 200);

/* Размер TOC */
      *((__u8 *)&toc_data_len) = data_buff[1];
      *((__u8 *)&toc_data_len + 1) = data_buff[0];

/* Число записей TOC */
      toc_entries = (toc_data_len - 2)/11;

/* Число сессий */
      last = data_buff[3];
      printf("Число сессий на диске - %d\n", last);

      t = (toc_t *)malloc(toc_data_len);
      memset((void *)t, 0, toc_data_len);
      memcpy((void *)t, data_buff + 4, toc_data_len);

      free(data_buff);

      printf("Entry\tSession\tPoint\tMin\tSec\tFrame\tPMin\tPsec\tPFrame\tLBA\n");
      for(i = 0; i < toc_entries; i++) {
           printf("%d\t", i);
           printf("%d\t", (t + i)->snum);
           printf("%X\t", (t + i)->point);
           printf("%d\t", (t + i)->min);
           printf("%d\t", (t + i)->sec);
           printf("%d\t", (t + i)->frame);
           printf("%d\t", (t + i)->pmin);
           printf("%d\t", (t + i)->psec);
           printf("%d\t", (t + i)->pframe);

#define PMIN(i) (t + i)->pmin
#define PSEC(i) (t + i)->psec
#define PFRAME(i) (t + i)->pframe
#define MIN(i) (t + i)->min
#define SEC(i) (t + i)->sec
#define FRAME(i) (t + i)->frame
#define POINT(i) (t + i)->point

      if((POINT(i) == 0xC0) || (POINT(i) == 0xC1)) {
           printf("\n");
           continue;
        }

/* Координаты начала следующей возможной области программ содержит указатель B0
 * в полях Min/Sec/Frame. Переводим эти координаты в LBA-формат, с учетом того, что
 * перед треком находится Pre-gap область размером 150 секторов
 */
      if(POINT(i) == 0xB0) lba = MSF2LBA(MIN(i), SEC(i), FRAME(i)) + 150;
       else lba = MSF2LBA(PMIN(i), PSEC(i), PFRAME(i));
      printf("%u\n", lba);
     }

      free(t);
      return 0;
}
```

Пересчёт координат из MSF-формата в LBA выполняет уже знакомый нам макрос MSF2LBA:

```
      #define MSF2LBA(Min, Sec, Frame) (((Min * 60 + Sec) * 75 + Frame) - 150)
```

Теперь возьмём диск, на котором записано три сессии, и посмотрим на результат работы функции read_raw_toc():

```
Число сессий на диске - 3
Entry   Sess   Point   Min    Sec    Frame   PMin   Psec   PFrame   LBA
0       1      A0      0      0      0       1      0      0        4350
1       1      A1      0      0      0       1      0      0        4350
2       1      A2      0      0      0       0      50     18       3618
3       1      1       0      0      0       0      2      0        0
4       1      B0      3      20     18      79     59     74       15018
5       1      C0      102    0      156     97     32     10
6       1      C1      72     52     176     0      0      0
7       2      A0      0      0      0       2      0      0        8850
8       2      A1      0      0      0       2      0      0        8850
9       2      A2      0      0      0       3      43     35       16610
10      2      2       0      0      0       3      22     18       15018
11      2      B0      5      13     35      79     59     74       23510
12      3      A0      0      0      0       3      0      0        13350
13      3      A1      0      0      0       3      0      0        13350
14      3      A2      0      0      0       5      43     59       25634
15      3      3       0      0      0       5      15     35       23510
16      3      B0      7      13     59      79     59     74       32534
```

Указатель A0 третьей сессии (запись 12) содержит в поле PMin номер первого трека последней сессии. Это значение равно 3. Запись 15 в полях PMin/PSec/PFrame содержит стартовые координаты этого трека - 5/15/35, или 23510 в LBA-формате. Это и есть искомое число A. Координаты начала следующей возможной области программ – значение числа B - содержит указатель B0 в полях Min/Sec/Frame (запись 16) – 7/13/59, или 32534 в LBA формате.

Второй способ определения значения числа A - чтение информации о сессиях. Для этого в поле Format командного блока READ TOC/PMA/ATIP (см. [1], табл. 441) должно принять значение 1. Блок информации о сессиях имеет вид, представленный на рис.41:

READ TOC/PMA/ATIP response data (Format = 0001b): Multi-session Information

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB) TOC Data Length							
1								(LSB)
2	First Complete Session Number							
3	Last Complete Session Number							
	TOC Track Descriptor							
0	Reserved							
1	ADR				CONTROL			
2	First Track Number In Last Complete Session							
3	Reserved							
4	(MSB)							
5	Start Address of First Track in Last Session							
6								
7								(LSB)

Рис.41. Блок информации о сессиях

Здесь First и Last Complete Session Number - номера первой и последней завершенной сессии, First Track Number In Last Complete Session - номер первого трека в последней завершенной сессии, Start Address of First Track in Last Session - стартовый адрес этого трека. Формат блока информации о сессиях описывает следующая структура:

```
typedef struct {
      __u8 res1;
      __u8 ctrl      :4;
      __u8 adr       :4;
      __u8 first_trk; // номер первого трека посл. сессии
      __u8 res2;
      __u32 start_addr; // стартовый адрес трека
}  ms_info_t;
```

Чтение блока информации о сессиях выполняет функция read_ms_info():

```
void read_ms_info()
{
      int first_trk = 0;  // номер первого трека посл. сессии
      __u8 read_toc_cmd[10]; // CDB - Command Descriptor Block
      __u8 data_buff[12];
      __u32 start_addr = 0; // стартовый адрес трека
      ms_info_t *ms_info;

      memset(data_buff, 0, 12);
      ms_info = (void *)(data_buff + 4);

/* Формируем командный пакет */
      memset(read_toc_cmd, 0, 10);
      read_toc_cmd[0] = READ_TOC; // код команды
      read_toc_cmd[2] = 1; // Format = 1, Multi-session Information
      read_toc_cmd[8] = 12; // размер блока данных

      send_cmd(sg_fd, read_toc_cmd, 10, SG_DXFER_FROM_DEV, data_buff, 12, 20);

      first_trk = ms_info->first_trk;
      start_addr = __swab32(ms_info->start_addr);

      printf("Первый трек последней сессии - %d\n", first_trk);
      printf("Стартовый адрес трека - %u\n", start_addr);
      return;
}
```

Стартовый адрес следующей возможной области программ (наше число B) также является адресом невидимого трека (invisible track), и определить его можно при помощи команды READ TRACK INFORMATION. Формат этой команды представлен на рис.42:

READ TRACK INFORMATION CDB

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (52h)							
1	Reserved						Address/Number Type	
2	(MSB)							
3				Logical Block Address/				
4				Track/Session Number				
5								(LSB)
6	Reserved							
7	(MSB)			Allocation Length				
8								(LSB)
9	Control Byte							

Рис.42. Формат команды READ TRACK INFORMATION

Значение поле Address/Number Type определяет содержание поля Logical Block Address/Track/Session Number. Если Address/Number Type = 01b, то устройство вернет блок информации о треке, номер которого находится в поле Logical Block Address/Track/Session Number. Если поле Track Number содержит значение 0xFF, то будет прочитана информация о невидимом (незавершенном) треке.

Формат блока информации о треке представлен на рис.43.

Track Information Block

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)			Data Length				
1								(LSB)
2	Track Number (Least Significant Byte)							
3	Session Number (Least Significant Byte)							
4	Reserved							
5	Reserved		Damage	Copy	Track Mode			
6	RT	Blank	Packet/Inc	FP	Data Mode			
7	Reserved						LRA_V	NWA_V

8	(MSB)
9	Track Start Address
10	
11	(LSB)
12	(MSB)
13	Next Writable Address
14	
15	(LSB)
16	(MSB)
17	Free Blocks
18	
19	(LSB)
20	(MSB)
21	Fixed Packet Size/ Blocking Factor
22	
23	(LSB)
24	(MSB)
25	Track Size
26	
27	(LSB)
28	(MSB)
29	Last Recorded Address
30	
31	(LSB)
32	Track Number (Most Significant Byte)
33	Session Number (Most Significant Byte)
34	Reserved
35	Reserved
36	(MSB)
……	Read Compatibility LBA
39	(LSB)

Рис.43. Формат блока информации о треке

Из всего многообразия данных, находящихся в блоке, нас интересует только поле Track Start Address. Это поле содержит стартовый адрес трека. Назначение остальных полей приведено в спецификации SCSI MMC ([1]).

Функция read_track_info считывает информацию о треке и определяет значение его стартового адреса. Входные парметры – номер трека:

```
__u32 read_track_info(int trk_num)
{
      __u8 read_track_info_cmd[10];
      __u8 data_buff[40];
      __u32 lba = 0;

      memset(data_buff, 0, 40);
      memset(read_track_info_cmd, 0, 10);
      read_track_info_cmd[0] = 0x52; // код команды
      read_track_info_cmd[1] = 1;
      read_track_info_cmd[5] = trk_num; // номер трека
      read_track_info_cmd[8] = 40;

      send_cmd(sg_fd, read_track_info_cmd, 10, SG_DXFER_FROM_DEV, data_buff, 40, 20);

/* Определяем стартовый адрес трека */
      memcpy((void *)&lba, (void *)(data_buff + 8), 4);
      return __swab32(lba);
}
```

В результате работы двух рассмотренных функций - read_ms_info() и read_track_info() - мы получим для имеющегося в нашем распоряжении трехсессионного диска значения чисел A и B: A = 23510 и B = 32534. Теперь можно сформировать ISO-образ для записи на диск четвертой сессии:

```
mkisofs -R -J -C 23510,32534 -M [имя файла устройства] -o track-04.iso [входной файл]
```

8.4 Чтение информации о диске

Перед тем, как приступить к записи информации, целесообразно выяснить, в каком состоянии находится диск, т.е. можно ли дописать на него новую сессию. Эту информацию можно получить при помощи команды READ DISK INFORMATION. Формат этой команды представлен на рис.44.

READ DISC INFORMATION CDB

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation Code (51h)							
1-6	Reserved							
7	(MSB) Allocation Length							
8								(LSB)
9	Control							

Рис.44. Формат команды READ DISK INFORMATION

Формат данных, возвращаемых устройством по этой команде, представлен на рис.45.

Disc Information Block

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB) Disc Information Length							
1								(LSB)
2	Reserved			Erasable	State of last Session		Disc Status	
3	Number of First Track on Disc							
4	Number of Sessions (Least Significant Byte)							
5	First Track Number in Last Session (Least Significant Byte)							
6	Last Track Number in Last Session (Least Significant Byte)							
7	DID_V	DBC_V	URU	DAC_V	Reserved	Dbit	BG Format Status	
8	Disc Type							
9	Number of Sessions (Most Significant Byte)							
10	First Track Number in Last Session (Most Significant Byte)							
11	Last Track Number in Last Session (Most Significant Byte)							
12	(MSB) Disc Identification							
13								
14								
15								(LSB)
16	(MSB) Last Session Lead-in Start Address							
17								
18								
19								(LSB)
20	(MSB) Last Possible Lead-out Start Address							
21								
22								
23								(LSB)
24	(MSB) Disc Bar Code							
...								
31								(LSB)
32	Disc Application Code							
33	Number of OPC Tables							
32-n	OPC Table Entries							

Рис.45. Блок информации о диске, возвращаемый по команде READ DISK INFORMATION

Необходимую нам информацию содержит поле Disk Status. Поле может принимать следующие значения:
- 00b - `Empty Disk`, диск пустой, не содержит информации
- 01b - `Incomplete Disk`, незавершенный диск, на который можно дописывать информацию
- 10b - `Finalized Disk`, диск завершен, информацию дописывать нельзя

Поле State of last Session описывает состояние последней сессии на диске. Поле может принимать следующие значения:
- 00b - пустая сессия `(Empty Session)`
- 01b - незавершенная сессия `(Incomplete Session)`
- 11b - сессия завершена `(Complete Session)`

Установленный в "1" бит Erasable свидетельствует о том, что в приводе находится перезаписываемый диск: CD/DVD-RW, DVD-RAM, DVD+RW.

Назначение остальных полей приведено в спецификации SCSI MMC-5, п. 6.27 "READ DISK INFORMATION Command" ([1]).

Чтение информации о диске выполняет функция read_disk_info():

```
int read_disk_info()
{
      __u8 read_disk_info_cmd[10];
      __u8 data_buff[34];

      memset(data_buff, 0, 34);
      memset(read_disk_info_cmd, 0, 10);
      read_disk_info_cmd[0] = 0x51;
      read_disk_info_cmd[8] = 34;

      send_cmd(sg_fd, read_disk_info_cmd, 10, SG_DXFER_FROM_DEV,\
           data_buff, sizeof(data_buff), 20);

      switch(data_buff[2] & 3) {

      case(0):
           printf("Диск пустой\n");
              break;

      case(1):
              printf("Можно дописывать информацию\n");
              break;

      case(2):
       printf("Запись невозможна\n");
              break;

       default:
              break;
      }

      return 0;
}
```

Если диск позволяет выполнить запись новой сессии (disk_status == 1), то алгоритм работы программы следующий:

- при помощи системного вызова stat определяется размер записываемого образа в блоках по 2048 байт (режим Data Mode 1);
- резервируется место для нового трека (команда RESERVE TRACK), при этом размер трека равен размеру файла-образа в блоках;
- при помощи команды READ TRACK INFORMATION определяется стартовый адрес зарезервированного трека, и, начиная с этого адреса, на диск выполняется запись данных.

Полный текст программы для записи многосессионного диска находится в файле SOURCE/SG/WRITE/TAO/DATA/iso2cd_multi.c.

8.5 Запись аудиоданных в режиме TAO

Алгоритм записи аудиоданных практически не отличается от приведенного ранее алгоритма записи данных. Но так как в одной сессии теперь будет находиться несколько треков, то при записи аудиотрека на компакт-диск необходимо будет знать стартовый адрес трека. Этот адрес можно вычислить самостоятельно, можно определить его как адрес невидимого, незавершенного трека (invisible, incomplete track), либо предварительно зарезервировать на компакт-диске пространство для каждого трека.

Рассмотрим пример программы для записи аудиоданных на диск с предварительным резервированием места для каждого трека. Входные параметры – имя каталога, содержащего файлы в MP3 формате.

```
int main(int argc, char **argv)
{
   int i = 1;
   struct stat s;
   struct dirent *d;
   DIR *dp;
   __u8 full_path[255];
   __u32 start_lba = 0, track_size = 0;

/* Проверяем наличие входных параметров */
   if(argc != 2) {
       printf("\n\tUsage: mp2cdda [Dir. with MP3-files]\n\n");
       return 0;
   }

/* Открываем файл устройства */
   if((sg_fd = open(SG_DEV, O_RDWR)) < 0) {
        perror("open");
        return -1;
   }

/* Устанавливаем параметры режима записи:
 * - тип блока данных dbt = 0 - "сырые" данные, размер блока 2352 байта
 * - track_mode = 0 - трек содержит аудиоданные
 */
   if(mode_select(0, 0) < 0) return 0;

/* Открываем каталог с нашими файлами */
   if((dp = opendir(argv[1])) == NULL) {
       perror("opendir");
       return -1;
   }

/* Пропускаем записи родительского и текущего каталогов */
   d = readdir(dp); // "."
   d = readdir(dp); // ".."

   for(i = 1;; i++) {

       /* Считываем записи каталога */
       if((d = readdir(dp)) == NULL) break;
       printf("\n%d. %s\n", i, d->d_name);

       /* Формируем полное имя файла */
       memset(full_path, 0, 255);
       sprintf(full_path, "%s/%s", argv[1], d->d_name);

       /* Декодируем файл из MP3 формата в WAV */
       printf("\nDecoding file %s..\n", d->d_name);
       mp3_decoder(full_path);
       printf("OK\n");

       /* Определяем размер файла в блоках по 2352 байта */
       memset((void *)&s, 0, sizeof(struct stat));
       if(stat("./track.wav", &s)) { perror("stat"); return -1; }
       track_size = (s.st_size - 44)/ CD_FRAMESIZE_RAW;

       /* Резервируем место для трека. Если места для трека недостаточно (слишком большой файл)
        * переходим к следующему файлу в каталоге
        */
       if(reserv_track(track_size) < 0) {
              printf("Cannot reserve track %d\n", i);
              continue;
       }

       /* Считываем содержимое PMA области диска */
       read_pma();

       /* Определяем стартовый адрес зарезервированного трека */
       start_lba = read_track_info(i);
```

```
        printf("Track number: %d\n", i);
        printf("Track size: %u\n", track_size);
        printf("Start LBA of track #%d: %u\n", i, start_lba);

        /* Записываем трек на диск */
        if(write_audio("./track.wav", start_lba) < 0) {
            printf("Cannot write %s\n", d->d_name);
            return -1;
        }

        /* Закрываем трек и переходим к следующему файлу в каталоге */
        if(close_track(i) < 0) {
               printf("Cannot close track #%d!\n", i);
               return -1;
        }
    }

/* Закрываем сессию */
    fprintf(stdout, "Close session..\n");
    if(close_session() < 0) printf("\nCannot close session!\n");
    else printf("OK\n\n");

/* Закрываем каталог, удаляем файл track.wav и открываем лоток */
    closedir(dp);
    unlink("./track.wav");
    eject_cd();
    close(sg_fd);
    return 0;
}
```

Декодирование файла из MP3 формата в WAV выполняет функция mp3_decoder:

```
void mp3_decoder(__u8 *file_name)
{
   static pid_t pid;
   int status;

   switch(pid = fork()) {

case -1:
    perror("fork");
    exit(-1);

case 0:
    execl("/usr/bin/mpg321", "mpg321", "-q", "-w", "track.wav", file_name, 0);
    exit(-1);
   }

   if((pid = waitpid(pid, &status, 0)) && WIFEXITED(status)) return;
}
```

Входной параметр функции – имя файла в MP3 формате. Для декодирования используется утилита mpg321.

Полный текст программы для записи Audio-CD находится в файле
SOURCE/SG/WRITE/TAO/AUDIO/mp2cdda_reserv_track.c.

Приведем пример работы этой программы. Устанавливаем в привод CD-RW диск и запускаем программу на выполнение, указав в параметрах имя каталога, содержащего три файла в MP3 формате:

```
       # ./mp2cdda_reserv_track  /tmp/MUSIC
```

В результате получаем:

```
1. 12 - Ведьма и Осел.mp3

Decoding file 12 - Ведьма и Осел.mp3..
OK
PMA data length: 26
PMA entries: 2
```

```
Entry   ADR     CTRL    Point   Zero    Min     Sec     Frame   PMin    Psec    PFrame  LBA
0       2       0       0       0       49      0       99      0       0       0       ---
1       1       0       1       0       2       59      55      0       2       0       0
Track number: 1
Track size: 13328
Start LBA of track #1: 0
LBA:  13328 ( 13328)

2. 03 - Бедняжка.mp3

Decoding file 03 - Бедняжка.mp3..
OK
PMA data length: 37
PMA entries: 3
Entry   ADR     CTRL    Point   Zero    Min     Sec     Frame   PMin    Psec    PFrame  LBA
0       2       0       0       0       49      0       99      0       0       0       ---
1       1       0       1       0       2       59      55      0       2       0       0
2       1       0       2       0       7       11      72      3       1       55      13480
Track number: 2
Track size: 18765
Start LBA of track #2: 13480
LBA:  18765 ( 32245)

3. 01 - Король и Шут.mp3

Decoding file 01 - Король и Шут.mp3..
OK
PMA data length: 48
PMA entries: 4
Entry   ADR     CTRL    Point   Zero    Min     Sec     Frame   PMin    Psec    PFrame  LBA
0       2       0       0       0       49      0       99      0       0       0       ---
1       1       0       1       0       2       59      55      0       2       0       0
2       1       0       2       0       7       11      72      3       1       55      13480
3       1       0       3       0       9       55      35      7       13      72      32397
Track number: 3
Track size: 12111
Start LBA of track #3: 32397
LBA:  12111 ( 44508)
```

Хорошо видно, что после каждого вызова функции reserv_track в PMA добавляется запись о новом треке. Расстояние между треками – 2 секунды.

Теперь рассмотрим пример записи аудиоданных без предварительного резервирования треков. Для определения стартового адреса трека при помощи команды READ_TRACK_INFORMATION считывается адрес невидимого (invisible) трека. В этом случае поле Address/Number Type команды READ TRACK INFORMATION (см. рис.42) содержит 01b, поле Logical Block Address/Track/Session Number - значение 0xFF.

```
int main(int argc, char **argv)
{
   int i = 1;
   struct stat s;
   struct dirent *d;
   DIR *dp;
   __u8 full_path[255];
   __u32 start_lba = 0, track_size = 0;

/* Проверяем наличие входных параметров – имя каталога с файлами в MP3 формате */
   if(argc != 2) {
       printf("\n\tUsage: mp2cdda [Dir. with MP3-files]\n\n");
       return 0;
   }

/* Открываем файл устройства */
   if((sg_fd = open(SG_DEV, O_RDWR)) < 0) {
       perror("open");
       return -1;
   }

/* Устанавливаем параметры режима записи: записываем аудиоданные, размер сектора – 2352 байт */
   if(mode_select(0, 0) < 0) return 0;
```

```
/* Открываем каталог с файлами */
    if((dp = opendir(argv[1])) == NULL) {
        perror("opendir"); return -1;
    }

/* Пропускаем записи родительского и текущего каталогов */
    d = readdir(dp); // "."
    d = readdir(dp); // ".."

    for(i = 1;; i++) {

        /* Считываем записи каталога */
        if((d = readdir(dp)) == NULL) break;
        printf("\n%d. %s\n", i, d->d_name);

        /* Формируем путь к файлу */
        memset(full_path, 0, 255);
        sprintf(full_path, "%s/%s", argv[1], d->d_name);

        /* Декодируем файл из MP3 формата в WAV */
        mp3_decoder(full_path);

        /* Определяем размер файла в блоках по 2352 байта */
        memset((void *)&s, 0, sizeof(struct stat));
        if(stat("./track.wav", &s)) { perror("stat"); return -1; }
        track_size = (s.st_size - 44)/ CD_FRAMESIZE_RAW;

        /* Определяем стартовый адрес невидимого трека */
        start_lba = read_track_info(0xFF);
        printf("Track number: %d\n", i);
        printf("Track size: %u\n", track_size);
        printf("Start LBA of track #%d: %u\n", i, start_lba);

        /* Записываем аудиоданные в трек */
        if(write_audio("./track.wav", start_lba) < 0) return -1;

        /* Закрываем трек */
        if(close_track(i) < 0) return -1;
    }

/* По окончании записи всех треков закрываем сессию, каталог, удаляем файл track.wav и извлекаем диск из лотка */
    close_session();
    closedir(dp);
    unlink("./track.wav");
    eject_cd();

    close(sg_fd);
    return 0;
}
```

Полный текст программы находится в файле SOURCE/SG/WRITE/TAO/AUDIO/mp2cdda_inv_track.c.

Модифицируем эту программу для возможности создания Audio-CD из файлов формата OggVorbis. Нам понадобится новая функция декодирования файла из формата OggVorbis в WAV:

```
void ogg_decoder(__u8 *file_name)
{
    __u8 pcmout[2352];
    FILE *f;
    int out;
    OggVorbis_File vf;
    int current_section;

    f = fopen(file_name,"r");
    out = open("./track.cdr", O_CREAT|O_RDWR|O_TRUNC, 0600);

    if(ov_open(f, &vf, NULL, 0) < 0) {
        fprintf(stderr,"Input does not appear to be an Ogg bitstream.\n");
        exit(-1);
```

```
    }

    for(;;) {
        memset(pcmout, 0, 2352);
        long ret = ov_read(&vf, pcmout, sizeof(pcmout), 0, 2, 1, &current_section);
        if (ret == 0) break;
        if(ret < 0) exit(-1);
        if(write(out, pcmout, ret) < 0) exit(-1);
    }

    fclose(f);
    close(out);

    return;
}
```

Преобразование файла из формата Ogg Vorbis в WAV выполняет функция ogg_decoder(). Входной параметр функции - имя файла в формате Ogg Vorbis. Для конвертирования используется библиотека libvorbis (http://www.vorbis.org). На выходе получается файл track.cdr в формате WAV, но без RIFF-заголовка.

Полный текст программы находится в файле SOURCE/SG/WRITE/TAO/AUDIO/ogg2cdda_inv_track.c.

Стартовый адрес нового трека также можно вычислить самостоятельно по следующей формуле:

```
start_address = total_sectors + apl + 2,
```

где total_sector - общее число секторов, записанных на диск, apl - длина аудиопаузы, значение поля Audio Pause Length (APL) страницы параметров режима записи. Значение APL хранится в поле apl структуры wpm, и определяется следующим образом:

```
apl = __swab16(wpm->apl);
```

В файле SOURCE/SG/WRITE/TAO/AUDIO/mp2cdda.c находится полный текст программы, выполняющей запись MP3-файлов на аудио-компакт. Стартовый адрес трека вычисляется вышеприведенным способом.

8.6 Особенности записи информации в режиме SAO

В отличии от режима записи Track-at-once (TAO), в режиме Session-at-once (SAO) треки примыкают друг к другу вплотную, промежутки (паузы) между ними отсутствуют. Для управления процессом записи устройству передается специальная структура, содержащая информацию о расположении (координатах) треков, форматах блоков основного канала и субканалов трека. Эта структура, которая называется CUE SHEET, является своего рода картой, на основании которой устройство сформирует входную (Lead-In) и входную (Lead-Out) области сессии (диска). Структура CUE SHEET состоит из последовательности 8-байтовых блоков, самый первый блок описывает входную область сессии, последний - выходную область, остальные блоки содержат информацию о треках.

Блоки имеют следующий формат:

1	2	3	4	5	6	7	8
CTL/ADR	TNO	INDEX	DATA FORM	SCMS	ABSOLUTE TIME		

Рис.46. Формат блока структуры CUE SHEET

Назначение полей:
1. CTL/ADR - значение байта CTL/ADR трека.
2. TNO - номер трека
3. INDEX - индекс трека
4. DATA FORM - формат данных трека
5. SCMS - байт системы управления копированием
6. ABSOLUTE TIME - стартовые координаты трека в MSF-формате

Формат байта CTL/ADR представлен на рис.47. Старшие 4 разряда байта CTL/ADR занимает поле управления CTL, младшие 4 разряда - поле ADR.

7	6	5	4	3	2	1	0
CTL Field				ADR Field			

Рис.47. Байт CTL/ADR

Поле CTL определяет тип информации, находящейся в треке, и может принимать следующий значения:
- 00xxb - 2 аудиоканала
- 10xxb - 4 аудиоканала
- 01xxb - трек данных
- 11xxb - зарезервировано

Для Lead-In и Lead-Out областей значение байта CTL/ADR должно быть равно 0x01 (за исключением случая, когда поле Data Form = 0x41).

Байт TNO и INDEX для Lead-In области принимают значение 0x00. Lead-Out область, согласно спецификации, кодируется как трек под номером 0xAA, поле INDEX всегда равно 0x01.

Структура байта DATA FORM показана на рис.48:

7	6	5	4	3	2	1	0
Data Form of Sub-Channel		Data Form of Main Data					

Рис.48. Структура поля DATA FORM

Поле Data Form of Main Data определяет формат блоков данных основного канала - CD-DA (аудиоданные), CD-ROM Mode 1, CD-ROM XA. При записи аудио это поле принимает значение 0x00, и устройству передается блок аудиоданных размером 2352 байта. При записи данных в формате Data Mode 1 (другие форматы в данном документе не рассматриваются) поле Data Form of Main Data может принимать следующие значения:
- 0x10 - приложение передает устройству блок данных User Data размером 2048 байт, поля Sync/Header и EDC/ECC генерируются устройством;
- 0x11 - приложение передает устройству блок данных размером 2352 байт в составе полей User Data (2048 байт), Sync/Header (16 байт) и EDC/ECC (288 байт), однако устройство содержимое полей Sync/Header и EDC/ECC игнорирует и генерирует собственные значения.

Поле Data Form of Sub-Channel определяет формат данных субканалов, передаваемых устройству. Согласно спецификации SCSI MMC-5 [1], данные субканалов P и Q, переданные в структуре CUE SHEET, устройство игнорирует и генерирует собственные значение. Формат поля Data Form of Sub-Channel определен в спецификации SCSI MMC-5, табл. 516.

После того, как структура CUE SHEET сформирована, она передается устройству при помощи команды SEND CUE SHEET. Формат этой команды представлен на рис.49:

SEND CUE SHEET CDB

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation Code (5Dh)							
1-6	Reserved							
7	(MSB) CUE SHEET SIZE							
8								(LSB)
9	Control							

Рис.49. Формат команды SEND CUE SHEET

Параметр Cue Sheet Size содержит размер структуры CUE SHEET в байтах.

8.7 Запись данных на компакт-диск в режиме SAO

Рассмотрим пример программы, выполняющей запись односессионного CD-R/RW диска в режиме SAO. В сессии находится только один трек, формат данных - CD-ROM Mode 1. Алгоритм работы программы следующий:
- проверяем возможность записи информации в режиме SAO;
- в странице параметров режима записи указываем требуемый режим записи - SAO: Write Type = 0x02;
- формируем структуру CUE SHEET и отправляем ее устройству;

- выполняем запись данных на носитель.

Для возможности записи в режиме SAO устройство должно обладать свойством CD Mastering (код 0x002E). Формат дескриптора свойства CD Mastering приведен на рис.50:

CD Mastering Feature Descriptor Format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB) Feature Code = 002Eh							
1								(LSB)
2	Reserved		Version				Persistent	Current
3	Additional Length = 04h							
4	Resvd	BUF	SAO	Raw MS	Raw	Test Write	CD-RW	R-W
5	(MSB) Maximum CUE SHEET Length							
6								
7								(LSB)

Рис.50. Формат дескриптора свойства CD Mastering

Устройство поддерживает режим записи SAO, если бит SAO установлен в единицу. Формат дескриптора свойства можно описать при помощи следующей структуры:

```
/* Свойство CD Mastering */
typedef struct {
    __u16 f_code;
    __u8 current :1;
    __u8 persistent :1;
    __u8 version :4;
    __u8 res1 :2;
    __u8 add_length;
    __u8 rw :1;
    __u8 cd_rw :1;
    __u8 test_write :1;
    __u8 raw :1;
    __u8 raw_ms :1;
    __u8 sao :1;
    __u8 BUF :1;
    __u8 res2 :1;
    __u8 max_cue_len[3];
}  cd_sao_t;
```

Проверка наличия свойства выполняет при помощи команды GET CONFIGURATION (см. пункт "Свойства и профили устройства"). Проверку выполняет функция check_feature(). Входные параметры функции - код проверяемого свойства, в нашем случае это 0x002E:

```
int check_feature(__u16 f_num)
{
    __u8 get_conf_cmd[10]; // CDB
    __u8 data_buff[FEATURE_LEN]; // результаты чтения
    __u32 data_length = 0;  // реальная длина данных
    __u16 cur_prof = 0; // текущий профиль, Current Profile

    cd_sao_t *cd_sao;

    memset(data_buff, 0, sizeof(data_buff));
    cd_sao = (void *)(data_buff + 8);

/* Для определения свойств устройства используется команда GET CONFIGURATION */
    memset(get_conf_cmd, 0, 10);
    get_conf_cmd[0] = 0x46; // код команды GET CONFIGURATION
    get_conf_cmd[1] = 2; // RT = 10b, считываем только запрашиваемое свойство
    get_conf_cmd[2] = *((__u8 *)&f_num + 1);
    get_conf_cmd[3] = *((__u8 *)&f_num);
    get_conf_cmd[8] = 16;

/* Посылаем устройству команду */
    test_unit_ready(sg_fd);
```

```
    send_cmd(sg_fd, get_conf_cmd, 10, SG_DXFER_FROM_DEV, data_buff, 16, 200);

    memcpy((void *)&data_length, data_buff, 4);
    data_length = __swab32(data_length);
    if(data_length == 4) return -1; // свойство не поддерживается

/* Проверяем, является ли свойство текущим */
    if(cd_sao->current != 1) return -1;

/* Определяем текущий профиль */
    *((__u8 *)&cur_prof) = data_buff[7];
    *((__u8 *)&cur_prof + 1) = data_buff[6];
    printf("\nТекущий профиль - 0x%04X\n", cur_prof);

/* Возвращаем значение бита SAO */
    return (cd_sao->sao);
}
```

Формирование структуры CUE SHEET выполняет функция send_cue_sheet. По условиям задачи, на диск записывается сессия, состоящая из одного трека, формат данных - CD-ROM Mode 1. Структура CUE SHEET будет состоять из четырех 8-байтовых блоков, и ее размер равен 32 байта (4 x 8). Первый и последний блоки описывают входную и выходную области сессии, второй блок - Pre-Gap область трека, третий блок описывает сам трек.

Для указания стартовых координат (поле ABSOLUTE TIME) необходимо выполнить преобразование адреса из формата LBA в MSF. Для этого спецификацией SCSI MMC-5 [1] (табл.584 «LBA to MSF translation») предусмотрены следующие формулы:

```
    M = IP((LBA + 150) / (60 * 75))
    S = IP((LBA + 150 - M * 60 * 75) / 75)
    F = IP(LBA + 150 - M * 60 * 75 - S * 75)
```

Здесь IP - это Integer Part (целая часть), параметр LBA принимает значение -151 < LBA < 404850.

Пересчет координата из LBA в MSF выполняется при помощи следующих макроопределений:

```
#define LBA2MIN(LBA) ((LBA + 150) / (60 * 75))
#define LBA2SEC(LBA, MIN) ((LBA + 150 - (MIN * 60 * 75)) / 75)
#define LBA2FRAME(LBA, MIN, SEC) ((LBA + 150 - (MIN * 60 * 75)) - (SEC * 75))
```

В параметрах функции send_cue_sheet передается размер трека, исчисляемый в блоках по 2048 байт.

```
int send_cue_sheet(__u32 trk_size)
{
    __u8 cue_sheet_cmd[10];
    __u8 cue_buff[32]; // структура CUE SHEET, размер 32 байта
    __u8 min, sec, frame; // координаты Lead-Out области

    memset(cue_sheet_cmd, 0, 10);
    memset(buff, 0, 32);

/* Заполняем поля CUE SHEET. Начинаем с Lead-In области */
    cue_buff[0] = 0x01; // CTL/ARD
    cue_buff[1] = 0x00; // номер трека, всегда 0
    cue_buff[2] = 0x00; // индекс трека, всегда 0
    cue_buff[3] = 0x01; // DATA FORM, всегда 1
    cue_buff[4] = 0x00; // SCMS
    cue_buff[5] = 0x00; // Minute = 0
    cue_buff[6] = 0x00; // Second = 0
    cue_buff[7] = 0x00; // Frame = 0

/* Pre-gap область первого трека */
    cue_buff[8] = 0x41; // CTL = 4 - трек содержит данные
    cue_buff[9] = 0x01; // номер трека = 1
    cue_buff[10] = 0x00; // индекс трека = 0
    cue_buff[11] = 0x10; // данные в формате Mode 1
    cue_buff[12] = 0x00; // SCMS
```

```
/* Координаты Pre-Gap области:  (–150) в формате LBA,
 * или 0/0/0 в MSF формате
 */
    cue_buff[13] = 0x00; // Minute
    cue_buff[14] = 0x00; // Second
    cue_buff[15] = 0x00; // Frame

/* Первый трек */
    cue_buff[16] = 0x41; // CTL = 4 - трек содержит данные
    cue_buff[17] = 0x01; // номер трека = 1
    cue_buff[18] = 0x01; // индекс трека = 1
    cue_buff[19] = 0x10; // данные в формате Mode 1
    cue_buff[20] = 0x00; // SCMS

/* Координаты первого трека – 0 в формате LBA,
 * или 0/2/0 в MSF формате
 */
    cue_buff[21] = 0x00; // Minute
    cue_buff[22] = 0x02; // Second
    cue_buff[23] = 0x00; // Frame

/* Lead-Out */
    cue_buff[24] = 0x01; // CTL/ADR
    cue_buff[25] = 0xAA; // номер трека выходной области
    cue_buff[26] = 0x01; // индекс, для Lead-Out всегда 1
    cue_buff[27] = 0x01; // DATA FORM, всегда 1
    cue_buff[28] = 0x00; // SCMC

/* Определяем координаты Lead-Out области. Эта область расположена сразу за первым
 * треком. Размер трека нам известен
 */
    min = LBA2MIN(trk_size);
    sec = LBA2SEC(trk_size, min);
    frame = LBA2FRAME(trk_size, min, sec);
    printf("\nMIN/SEC/FRAME: %.2d/%.2d/%.2d\n", min, sec, frame);

    cue_buff[29] = min;
    cue_buff[30] = sec;
    cue_buff[31] = frame;

/* Формируем команду SEND CUE SHEET */
    cue_sheet_cmd[0] = 0x5D;
    cue_sheet_cmd[8] = 32;

/* Посылаем устройству команду */
    if(send_cmd(cue_sheet_cmd, 10, SG_DXFER_TO_DEV, cue_buff, 32, 200) < 0) return -1;
    return 0;
}
```

После того, как структура CUE SHEET отправлена устройству, можно приступить к записи данных на носитель. Запись выполняет функция write_iso, параметр функции - имя файла-образа. Вначале записывается Pre-Gap область первого трека, а затем сам трек. Стартовый адрес Pre-Gap области в формате MSF равен 0/0/0 (-150 в LBA формате), размер Pre-Gap области составляет 150 секторов.

```
int write_iso(__u8 *file_name)
{

#define PREGAP_SIZE 307200 // размер Pre-Gap области: 150 секторов по 2048 байт

    int ret, in_f;
    __u8 write_cmd[10];
    __u8 *write_buff;
    int lba, lba1 = -150;

    in_f = open(file_name, O_RDONLY);

    memset(write_cmd, 0, 10);
    write_cmd[0] = WRITE_10;
```

```
    write_cmd[8] = 150; // размер Pre-Gap в секторах

    write_buff = (__u8 *)malloc(PREGAP_SIZE);
    memset(write_buff, 0, PREGAP_SIZE);

/* Записываем Pre-gap область. Она начинается с сектора -150 и содержит нули */
    lba = __swab32(lba1);
    memcpy((write_cmd + 2), (void *)&lba, 4);
    lba1 += 150;

    ret = send_cmd(write_cmd, 10, SG_DXFER_TO_DEV, write_buff, PREGAP_SIZE, 200);
    if(ret < 0) return -1;

    free(write_buff);

/* Дальше выполняется запись данных первого трека аналогично
 * рассмотренным ранее примерам
 */
    . . . . . .
}
```

Полный текст программы для записи данных на CD-R/RW диск в режиме SAO находится в файле SOURCE/SG/WRITE/SAO/DATA/cue_data.c.

8.8 Запись аудиоданных в режиме SAO

При записи аудиоданных на компакт-диск в режиме SAO в одной сессии будет находиться несколько треков. Рассмотрим, как в этом случае формируется структура CUE SHEET, и для начала изучим формат этой структуры на реальном примере. С этой целью внесем небольшое дополнение в исходный код утилиты cdrecord, а именно в функцию fillcue (файл cdrecord/drv_mmc.c), в результате чего функция приобретает следующий вид:

```
LOCAL void
     fillcue(cp, ca, tno, idx, dataform, scms, mp)
     struct mmc_cue *cp; /* The target cue entry */
     int ca; /* Control/adr for this entry */
     int tno; /* Track number for this entry */
     int idx; /* Index for this entry */
     int dataform; /* Data format for this entry */
     int scms; /* Serial copy management */
     msf_t *mp; /* MSF value for this entry */
{
     cp->cs_ctladr = ca; /* XXX wie lead in */
     cp->cs_tno = tno;
     cp->cs_index = idx;
     cp->cs_dataform = dataform; /* XXX wie lead in */
     cp->cs_scms = scms;
     cp->cs_min = mp->msf_min;
     cp->cs_sec = mp->msf_sec;
     cp->cs_frame = mp->msf_frame;

     /* Этот код добавлен нами в учебных целях! */
     printf("\n0x%.2X\t", cp->cs_ctladr);
     printf("0x%.2X\t",cp->cs_tno);
     printf("0x%.2X\t",cp->cs_index);
     printf("0x%.2X\t",cp->cs_dataform);
     printf("0x%.2X\t",cp->cs_scms);
     printf("0x%.2X\t",cp->cs_min);
     printf("0x%.2X\t",cp->cs_sec);
     printf("0x%.2X\n",cp->cs_frame);
}
```

После внесения изменений пересоберем утилиту и запишем с ее помощью три аудиотрека:

```
    # cdrecord -dev=X,Y,Z -dao -pad -audio trk1.wav trk2.wav trk3.wav
```

Размер первого трека равен 17087 блоков, размер второго трека - 12923, размер третьего трека - 20580. Структура CUE SHEET имеет следующее содержание:

```
1    0x01    0x00    0x00    0x01    0x00    0x00    0x00    0x00 // Lead-In
```

```
2    0x01    0x01    0x00    0x00    0x00    0x00    0x00    0x00 // Pre-Gap первого трека
3    0x01    0x01    0x01    0x00    0x00    0x00    0x02    0x00 // Первый трек
4    0x01    0x02    0x00    0x00    0x00    0x03    0x2F    0x3E // Pre-Gap второго трека
5    0x01    0x02    0x01    0x00    0x00    0x03    0x31    0x3E // Второй трек
6    0x01    0x03    0x00    0x00    0x00    0x06    0x28    0x0A // Pre-Gap третьего трека
7    0x01    0x03    0x01    0x00    0x00    0x06    0x2A    0x0A // Третий трек
8    0x01    0xAA    0x01    0x01    0x00    0x0B    0x10    0x28 // Lead-Out
```

Структура CUE SHEET состоит из восьми блоков, ее размер равен 64 байт. Первый и последний блоки - это входная и выходная области сессии. Блок 2 - Pre-Gap область первого трека, стартовый адрес равен (-150). Третий блок описывает первый трек, стартовый адрес трека равен 0. Четвертый блок - Pre-Gap область второго трека, стартовый адрес области 16937. Пятый блок - второй трек, стартовый адрес трека равен размеру первого трека - 17087 блоков. Шестой блок - Pre-Gap область третьего трека, стартовый адрес равен 29860. Седьмой блок содержит описание третьего трека, его стартовый адрес равен 30010 (17087 + 12923). Параметр DATA FORM равен 0x00. Это означает, что на диск выполняется запись аудиоданных, и размер блока равен 2352 байта.

Можно, конечно, не париться с исходными текстами, а просто запустить cdrecord с ключем -vv и посмотреть содержимое области данных соответствующего командного пакета.

Таким образом, опытным путем установлено, что треки действительно расположены вплотную друг к другу, и Pre-Gap область трека захватывает последние 150 секторов области данных предыдущего трека. Исключение составляет Pre-Gap область первого трека.

Перепишем функцию send_cue_sheet для возможности записи на диск трех аудиотреков. Входные параметры функции - размеры треков в блоках по 2352 байта:

```
int send_cue_sheet(int trk1_size, int trk2_size, int trk3_size)
{
    __u8 cue_sheet_cmd[10];
    __u8 *cue_buff;
    __u8 min, sec, frame;
    int cb_size = 0;
    int trk = -150;

/* Определяем размер структуры CUE SHEET. В ее состав входят три Pre-Gap области,
 * три трека, входная и выходная области
 */
    cb_size = 3 * 8 + 3 * 8 + 16; // Итого 64 байта

    cue_buff = (__u8 *)malloc(cb_size);
    memset(cue_sheet_cmd, 0, 10);
    memset(cue_buff, 0, cb_size);

/* Формируем Lead-In область */
    cue_buff[0] = 0x01; cue_buff[1] = 0x00;
    cue_buff[2] = 0x00; cue_buff[3] = 0x01;
    cue_buff[4] = 0x00; cue_buff[5] = 0x00;
    cue_buff[6] = 0x00; cue_buff[7] = 0x00;

/* Pre-gap область первого трека */
    cue_buff[8] = 0x01; cue_buff[9] = 0x01;
    cue_buff[10] = 0x00; cue_buff[11] = 0x00;
    cue_buff[12] = 0x00; cue_buff[13] = 0x00;
    cue_buff[14] = 0x00; cue_buff[15] = 0x00;

/* Первый трек */
    cue_buff[16] = 0x01; cue_buff[17] = 0x01;
    cue_buff[18] = 0x01; cue_buff[19] = 0x00;
    cue_buff[20] = 0x00; cue_buff[21] = 0x00;
    cue_buff[22] = 0x02; cue_buff[23] = 0x00;

/* Pre-gap область второго трека захватывает последние
 * 150 секторов первого трека
 */
    trk += trk1_size;
    min = LBA2MIN(trk);
    sec = LBA2SEC(trk, min);
```

```
    frame = LBA2FRAME(trk, min, sec);

    cue_buff[24] = 0x01; cue_buff[25] = 0x02;
    cue_buff[26] = 0x00; cue_buff[27] = 0x00;
    cue_buff[28] = 0x00; cue_buff[29] = min;
    cue_buff[30] = sec; cue_buff[31] = frame;

/* Второй трек расположен сразу за первым, т.е. пауза
 * между треками отсутствует
 */
    sec += 2; // длина Pre-Gap – 2 секунды, или 150 секторов
    cue_buff[32] = 0x01; cue_buff[33] = 0x02;
    cue_buff[34] = 0x01; cue_buff[35] = 0x00;
    cue_buff[36] = 0x00; cue_buff[37] = min;
    cue_buff[38] = sec; cue_buff[39] = frame;

/* Pre-gap область третьего трека захватывает последние
 * 150 секторов второго трека
 */
    trk += trk2_size;
    min = LBA2MIN(trk);
    sec = LBA2SEC(trk, min);
    frame = LBA2FRAME(trk, min, sec);

    cue_buff[40] = 0x01; cue_buff[41] = 0x03;
    cue_buff[42] = 0x00; cue_buff[43] = 0x00;
    cue_buff[44] = 0x00; cue_buff[45] = min;
    cue_buff[46] = sec; cue_buff[47] = frame;

/* Третий трек */
    sec += 2;
    cue_buff[48] = 0x01; cue_buff[49] = 0x03;
    cue_buff[50] = 0x01; cue_buff[51] = 0x00;
    cue_buff[52] = 0x00; cue_buff[53] = min;
    cue_buff[54] = sec; cue_buff[55] = frame;

/* Lead-Out область */
    trk += trk3_size + 150;
    min = LBA2MIN(trk);
    sec = LBA2SEC(trk, min);
    frame = LBA2FRAME(trk, min, sec);

    cue_buff[56] = 0x01; cue_buff[57] = 0xAA;
    cue_buff[58] = 0x01; cue_buff[59] = 0x01;
    cue_buff[60] = 0x00; cue_buff[61] = min;
    cue_buff[62] = sec; cue_buff[63] = frame;

/* Формируем командный пакет */
    cue_sheet_cmd[0] = 0x5D;
    cue_sheet_cmd[8] = cb_size;

/* Отправляем устройству команду SEND_CUE_SHEET, освобождаем память
 * и выходим из функции
 */
    if(send_cmd(cue_sheet_cmd, 10, SG_DXFER_TO_DEV, \
                    cue_buff, cb_size, 200) < 0) return -1;
    free(cue_buff);
    return 0;
}
```

Отправив устройству структуру CUE SHEET, приступаем к записи данных.

Запись данных выполняет функция write_audio(). Входные параметры функции - имя файла в формате WAV и номер записываемого трека. Если записывается первый трек, то перед записью данных этого трека выполняется запись Pre-Gap области размером 150 секторов.

```
int write_audio(__u8 *file_name, int flag)
{

#define PREGAP_SIZE 352800 // 150 секторов по 2352 байта
```

```
      int i;
      int in_f;
      __u8 write_cmd[10];
      __u8 *write_buff;
      int lba = 0;
      static int lba1 = -150; // стартовый адрес для записи

      in_f = open(file_name, O_RDONLY);

      /* Перешагиваем через WAV-заголовок */
      lseek(in_f, 44, 0);

      memset(write_cmd, 0, 10);
      write_cmd[0] = WRITE_10;

      if(flag == 1) { // записываем первый трек

            write_buff = (__u8 *)malloc(PREGAP_SIZE);
            memset(write_buff, 0, PREGAP_SIZE);
            write_cmd[8] = 150;

      /* Записываем Pre-gap область первого трека */
            lba = __swab32(lba1);
            memcpy((write_cmd + 2), (void *)&lba, 4);
            lba1 += 150;
            if(send_cmd(write_cmd, 10, SG_DXFER_TO_DEV, write_buff, \
                        PREGAP_SIZE, 200)< 0) return 0;
            free(write_buff);
      }

/* Дальше выполняется запись данных аудиотрека */
    . . . . . .
}
```

Из всех Pre-Gap областей на диск записывается только Pre-Gap область первого трека, остальные просто обозначены в структуре CUE SHEET. Возникает вопрос - если не все Pre-Gap области подлежат записи, то зачем их указывать в CUE SHEET? Исключим "лишние" Pre-Gap области из структуры CUE SHEET, и функция send_cue_sheet примет следующий вид:

```
int send_cue_sheet(int trk1_size, int trk2_size, int trk3_size)
{
    __u8 cue_sheet_cmd[10];
    __u8 *cue_buff;
    __u8 min, sec, frame;
    int cb_size = 0;
    int trk = 0;

/* Вычисляем размер CUE SHEET - Pre-Gap область первого трека, три трека,
 * Lead-In и Lead-Out области
 */
    cb_size = 3 * 8 + 8 + 16; // Итого 48 байт

    cue_buff = (__u8 *)malloc(cb_size);
    memset(cue_sheet_cmd, 0, 10);
    memset(cue_buff, 0, cb_size);

/* Описание Lead-In области, Pre-gap области первого трека и непосредственно
 * первого трека точно такие же, как и в предыдущем примере.
 */

/* Lead-In область */
  .......

/* Pre-gap область первого трека */
  .......

/* Первый трек */
  .......
```

```
/* Второй трек */
    trk += trk1_size;
    min = LBA2MIN(trk);
    sec = LBA2SEC(trk, min);
    frame = LBA2FRAME(trk, min, sec);

    cue_buff[24] = 0x01; cue_buff[25] = 0x02;
    cue_buff[26] = 0x01; cue_buff[27] = 0x00;
    cue_buff[28] = 0x00; cue_buff[29] = min;
    cue_buff[30] = sec; cue_buff[31] = frame;

/* Третий трек */
    trk += trk2_size;
    min = LBA2MIN(trk);
    sec = LBA2SEC(trk, min);
    frame = LBA2FRAME(trk, min, sec);

    cue_buff[32] = 0x01; cue_buff[33] = 0x03;
    cue_buff[34] = 0x01; cue_buff[35] = 0x00;
    cue_buff[36] = 0x00; cue_buff[37] = min;
    cue_buff[38] = sec; cue_buff[39] = frame;

/* Lead-Out */
    trk += trk3_size;
    min = LBA2MIN(trk);
    sec = LBA2SEC(trk, min);
    frame = LBA2FRAME(trk, min, sec);

    cue_buff[40] = 0x01; cue_buff[41] = 0xAA;
    cue_buff[42] = 0x01; cue_buff[43] = 0x01;
    cue_buff[44] = 0x00; cue_buff[45] = min;
    cue_buff[46] = sec; cue_buff[47] = frame;

    cue_sheet_cmd[0] = 0x5D;
    cue_sheet_cmd[8] = cb_size;

    if(send_cmd(cue_sheet_cmd, 10, SG_DXFER_TO_DEV, \
                    cue_buff, cb_size, 200) < 0) return -1;

    free(cue_buff);
    return 0;
}
```

Перепишем функцию send_cue_sheet для возможности записи произвольного числа аудиотреков. Параметры функции - указатель на массив, содержащий имена файлов в формате WAV, и количество записываемых треков.

```
int send_cue_sheet(char **argv, int trk_num)
{
    int i = 0, n;
    __u8 cue_sheet_cmd[10];
    __u8 *cue_buff = NULL;
    __u8 min, sec, frame;
    struct stat s;
    int cb_size = 0;
    int trk = 0;

    cb_size = trk_num * 8 + 8 + 16;

    cue_buff = (__u8 *)malloc(cb_size);
    memset(cue_sheet_cmd, 0, 10);
    memset(cue_buff, 0, cb_size);

/* Lead-In область, Pre-gap первого трека, первый трек
 * (см. предыдущий пример)
 */
  .......

/* Записываем в структуру CUE SHEET информацию о треках.
 * Начинаем со второго трека, т.к. для первого запись уже сформирована
```

```
 */
    for(i = 1, n = 24; i < trk_num; i++, n += 8) {

/* Определяем размер трека */
           memset((void *)&s, 0, sizeof(struct stat));
           stat(argv[i], &s);

/* Стартовые координаты трека */
           trk += s.st_size/CD_FRAMESIZE_RAW;
           min = LBA2MIN(trk);
           sec = LBA2SEC(trk, min);
           frame = LBA2FRAME(trk, min, sec);

/* Формируем в структуре CUE SHEET запись для трека */
           cue_buff[n] = 0x01; cue_buff[n + 1] = i + 1;
           cue_buff[n + 2] = 0x01; cue_buff[n + 3] = 0x00;
           cue_buff[n + 4] = 0x00; cue_buff[n + 5] = min;
           cue_buff[n + 6] = sec; cue_buff[n + 7] = frame;
    }

/* Определяем размер последнего трека */
    memset((void *)&s, 0, sizeof(struct stat));
    stat(argv[trk_num], &s);

/* Формируем Lead-Out область */
    trk += s.st_size/CD_FRAMESIZE_RAW;
    min = LBA2MIN(trk);
    sec = LBA2SEC(trk, min);
    frame = LBA2FRAME(trk, min, sec);

    cue_buff[n] = 0x01; cue_buff[n + 1] = 0xAA;
    cue_buff[n + 2] = 0x01; cue_buff[n + 3] = 0x01;
    cue_buff[n + 4] = 0x00; cue_buff[n + 5] = min;
    cue_buff[n + 6] = sec; cue_buff[n + 7] = frame;

    cue_sheet_cmd[0] = 0x5D;
    cue_sheet_cmd[8] = cb_size;

    if(send_cmd(cue_sheet_cmd, 10, SG_DXFER_TO_DEV, \
              cue_buff, cb_size, 200) < 0) return -1;

    free(cue_buff);
    return 0;
}
```

Исходный текст программы для записи Audio-CD в режиме SAO находится в файле SOURCE/SG/WRITE/SAO/AUDIO/cue_audio.c.

9. Стирание информации с CD-RW диска

А теперь давайте вытрем с диска все, что мы на него записали. Для стирания информации с диска используется команда BLANK, формат которой приведен на рис.51:

BLANK CDB

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (A1h)							
1	Reserved			IMMED	Reserved	Blanking Type		
2	(MSB)							
3	Start Address/Track Number							
4								
5								(LSB)
6-10	Reserved							
11	Control Byte							

Рис.51. Формат команды BLANK

Поле Blanking type устанавливает режим очистки диска. Может принимать следующий значения:
- 000b - стирается вся информация, находящаяся на диске. Значение поля Start Address игнорируется

- 001b - минимальная очистка диска. Стирается PMA, Lead-In область диска и pre-gap область первого трека. Параметр Start Address игнорируется

Остальные значения поля Blanking type и их описание приведены в спецификации SCSI MMC-5, табл.210 "Blanking Types CD-RW" ([1]).

Перед тем, как послать устройству команду BLANK, необходимо проверить состояние бита Erasable блока информации о диске (рис.45). Если бит установлен в 1 – в приводе находится перезаписываемый диск, с которого можно стереть информацию.

Функция blank выполняет очистку CD-RW диска. Параметр функции blank_type устанавливает режим очистки диска. Допустимые значения этого параметра - 0 или 1:

```
int blank(__u8 blank_type)
{
      __u8 blank_cmd[12];
      memset(blank_cmd, 0, 12);
      blank_cmd[0] = 0xA1; // код команды BLANK
      blank_cmd[1] = blank_type; // режим очистки: 0 - полная, 1 - минимальная

      test_unit_ready(sg_fd);
      if(send_cmd(sg_fd, blank_cmd, 12, SG_DXFER_NONE, NULL, 0, 9600) < 0) return -1;
      return 0;
}
```

Полный текст программы очистки CD-RW диска приведен в файле SOURCE/SG/BLANK/blank.c.

10. DVD диск. Краткая история создания

В 1991 году компания PIONEER начал разработку нового видеодиска цифрового формата для возможности записи двух и более часов высококачественного видео на один диск, и в 1994 году на рынке была представлена производственная модель под названием Karaoke System, которая могла проигрывать 2.1 Гбайт видеоданных в формате MPEG-1 с одностороннего диска толщиной 1,2 мм. Для чтения использовался лазер с длиной волны 680 нм. В конце 1994 года PIONEER и TOSHIBA разрабатывают спецификацию на SD диск, использующий красный лазер. Приблизительно в это же время SONY и PHILIPS продвигают свою спецификацию – MMCD. Основным отличием этих спецификаций была толщина используемой подложки – в SD использовалось две подложки по 0,6 мм каждая, а в MMCD – одна 1,2 мм. В конце 1995 года две эти спецификации были объединены и создан DVD Консорциум. В августе 1996 года была опубликована спецификация DVD Video Book, и в ноябре того же года первые DVD плееры поступили в продажу. Так начал свою историю DVD диск.

11. Структура DVD-диска

11.1 Типы DVD-дисков

Главное преимущество, которым обладает DVD-диск по сравнению с CD-диском – это возможность хранения больших объемов информации, от 4 Гбайт и выше, в зависимости от типа диска. Существуют следующие типы DVD-дисков:

- DVD-ROM (read-only, только чтение);
- DVD-R, DVD+R (write-once, однократной записи);
- DVD-RAM, DVD-RW, DVD+RW (Re-rewritable, перезаписываемые).

DVD-диск может содержать информацию как на одной, так и на двух сторонах. В первом случае диск называется односторонним (Single Side), во втором случае – двусторонним (Double Side). На каждой стороне расположена информационная зона (Information Zone), представляющая собой спиральный трек, состоящий из трех областей – входной области (Lead-In Area), области данных (Data Area) и выходной области (Lead-Out Area). На диске типа DVD-ROM каждая сторона может содержать два информационных слоя. В случае двухслойного диска треки, принадлежащие разным слоям, могут быть расположены параллельно (Parallel Track Path), либо встречно (Opposite Track Path). При параллельном расположении каждый трек имеет собственную входную и выходную области. В случае встречного расположения треков входная и выходная область являются общими для каждого трека, и каждый трек имеет транзитную зону, которая называется серединной областью (Middle Area).

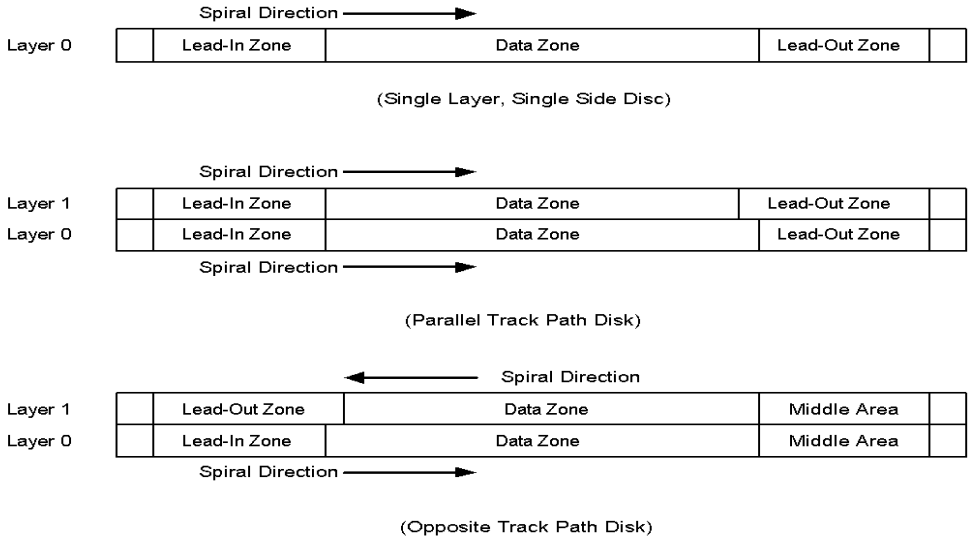

Рис.52. Типы DVD дисков

11.2 Формат физического сектора DVD-диска

В процессе записи данных на DVD-диск хост передает устройству блоки данных размером 2048 байт, называемые Main Data. Перед записью на диск блок Main Data проходит несколько этапов преобразования (промежуточных стадий преобразования). Рассмотрим последовательно эти этапы.

На первом этапе устройство из блока Main Data формирует блок Data Frame. Data Frame представляет собой матрицу 12 x 172, т.е. 12 строк по 172 байта в каждой (рис.53). Общий размер Data Frame равен 2046 байт (12 x 172 = 2046). В первой строке находятся три поля, называемые Identification Data (ID), ID Error Detection Code (IED) Copyright Management Information (CRP_MAI). Оставшиеся 160 байт первой строки и следующие 10 строк массива содержат данные Main Data. Последняя строка содержит 168 байт Main Data и 4 байта Error Detection Code (EDC). 2048 байт Main Data обозначены как $D_0 – D_{2047}$.

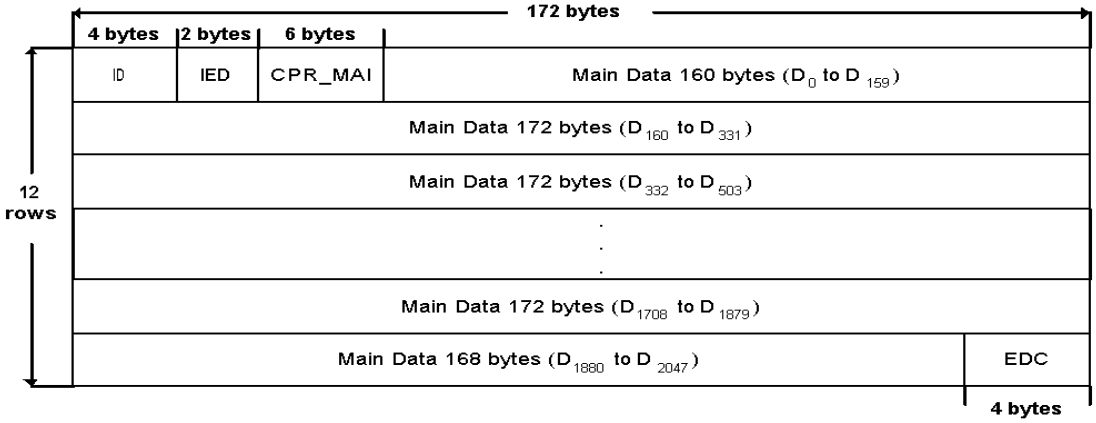

Рис.53. Формат Data Frame

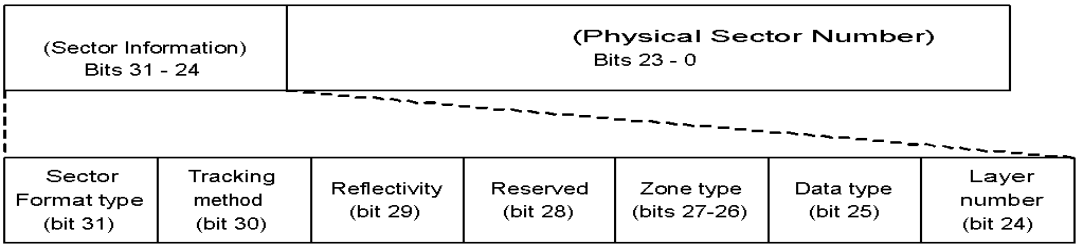

Рис.54. Формат поля ID

В поле Zone Type (биты 27-26) содержится информация о том, в какой области расположен сектор:

- 00b – в области данных

- 01b – в Lead-In области
- 10b – в Lead-Out области
- 11b – в Middle области

Значение бита Data type (бит 25) определяет тип данных сектора. Допустимые значения этого поля приведены в таблице 9:

Таблица 9. Значния бита Data type

Тип диска	Значение бита Data Type	
	0	1
ROM	Данные Read-Only	---
R	Данные Read-only	Следующий сектор принадлежит соединительной области (Linking data)*
-RW	Перезаписываемые данные (Re-recordable data)	Следующий сектор принадлежит соединительной области

**Термин «соединительная область» или «область потерь на связывание» будет рассмотрен ниже.*

Поле Physical Sector Number содержит физический номер сектора. DVD-диск использует двойную адресацию – PSN (Physical Sector Number) и LSN (Logical Sector Number, аналог LBA). PSN адресация применяется только на уровне внутреннего контроллера DVD-привода, LSN – на уровне внешней системы (компьютера). Адреса, заданные в этих форматах, связаны следующим соотношением: LSN = PSN - 0x30000

Блок Data Frame подвергается скремблированию, образуя Scrambled Frame. Детальное описание процесса скремблирования приведено в спецификации [Ecma-279].

16 последовательно расположеных Scrambled Frames образуют ECC блок, который можно представить в виде матрицы 192 x 172 (192 строки по 172 байта в каждой, рис.55). Для каждого столбца матрицы расчитывается и добавляется 16 байт корректирующего кода Reed-Solomon (208, 192, 17) (в спецификации [Ecma-279] обозначен как Parity of Outer Code, PO), тем самым матрица расширяется до 208 строк. Затем для каждой строки расчитывается и добавляется 10 байт корректирующего кода Reed-Solomon (182, 172, 11) (в спецификации [Ecma-279] обозначен как Parity of Inner Code, PI), и каждая строка расширяется до 182 байт.

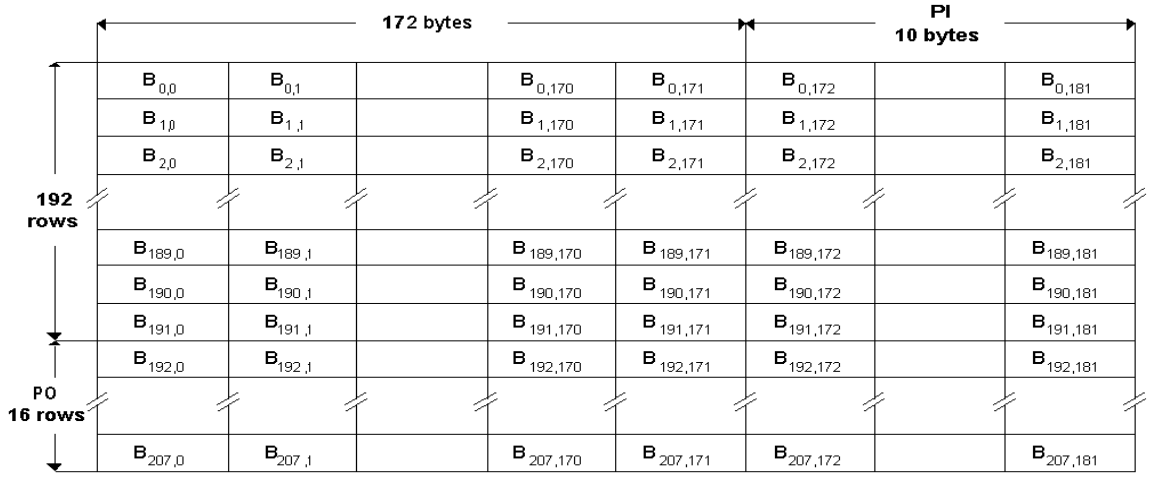

Рис.55. ECC блок

Далее из ECC блока формируется блок Recording Frame путем чередования строк матрицы следующим образом – строки, содержащие данные (первые 192 строки), чередуются со строками, содержащими корректирующий код (последние 16 строк) таким образом, что после 12 строк данных следует строка корректирующего кода. Таким образом 37856 байт ECC блока преобразуются в 16 блоков Recording Frames по 2366 байт каждый. Блок Recording Frame представляет собой матрицу 13 x 182 (13 строк по 182 байта в каждой, рис.56).

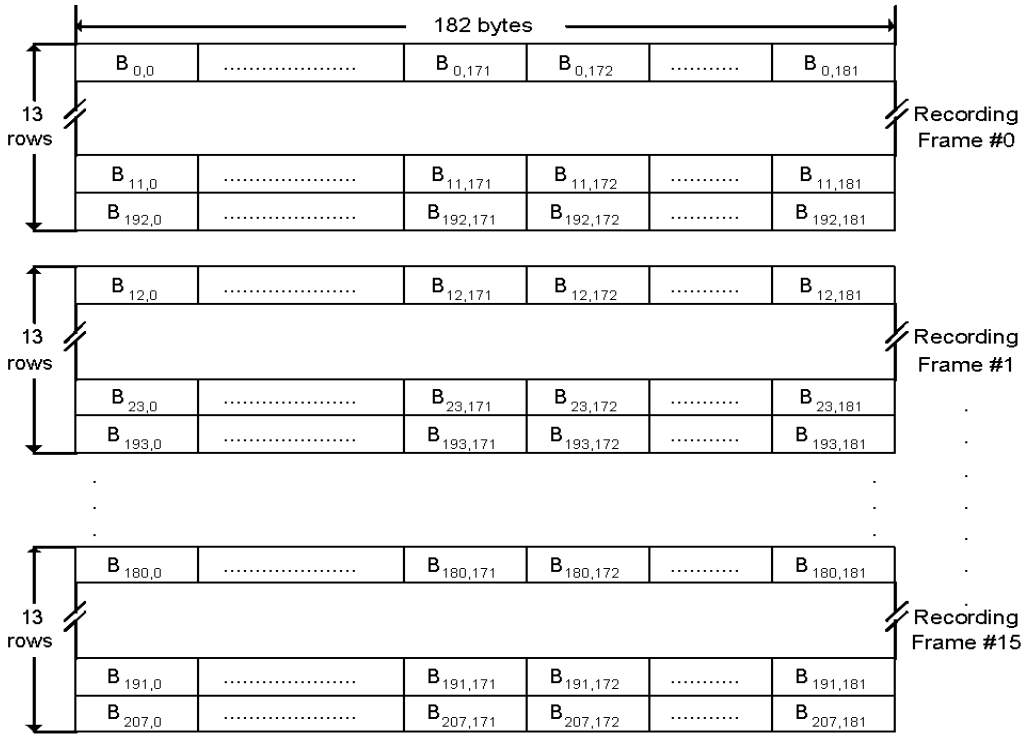

Рис.56. Recording Frames

8 бит каждого Recording Frame трансформируются в 16-ти битное кодовое слово (Code Words) таким образом, чтобы между двумя единичными битами было не меньше двух, но не более десяти нулевых бит. Этот код называется RLL(2,10), подробности кодирования приведены в приложении G спецификации ECMA-279 [14]. Затем кодовое слово конвертируется в 16 канальных битов (Channel bits) при помощи NRZI-конвертора (NRZI – Non Return to Zero Inverted, см. [ECMA-279]).

Промодулированный Recording Frame преобразуется в физический сектор (Physical Sector) путем разделения каждой строки на две равные части размером 91 байт (каждый байт расширяется до 16 бит за счет модуляции), и добавления к каждой части кода синхронизации SYNC Code. Формат Physical Sector показан на рис.57. Первая строка Recording Frame преобразуется в первую строку Physical Sector, вторая во вторую и т.д.

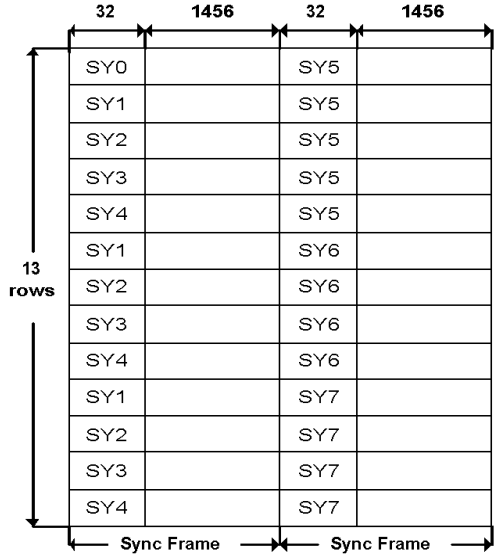

Рис.57. Формат Physical Sector

Physical Sector состоит из 13 строк, каждая строка содержит два Sync Frames в составе синхронизирующего кода SYNC Code и 1456-ти битной последовательности канальных битов Channel Bits (91 байт x 16 бит = 1456 бит). Запись физического сектора на диск начинается с первого Sync Frame, затем записывается второй и т.д.

11.3 Формат Lead-In области DVD-диска

Как известно, в Lead-In области компакт-диска (CD-ROM) находится таблица содержания диска (Table of Contains, TOC). Эта таблица используется программным обеспечением хоста (например, драйвером устройства) для получения информации о расположении треков**,** которые записаны на CD-диске. В отличии от CD, DVD-диск организован таким образом, что вся информация о данных находится внутри файловой системы, в области Data Zone, а Lead-In область содержит информацию, необходимую только самому устройству для корректной работы с DVD-диском. На рис.58 представлены форматы Lead-In областей DVD-дисков различных типов.

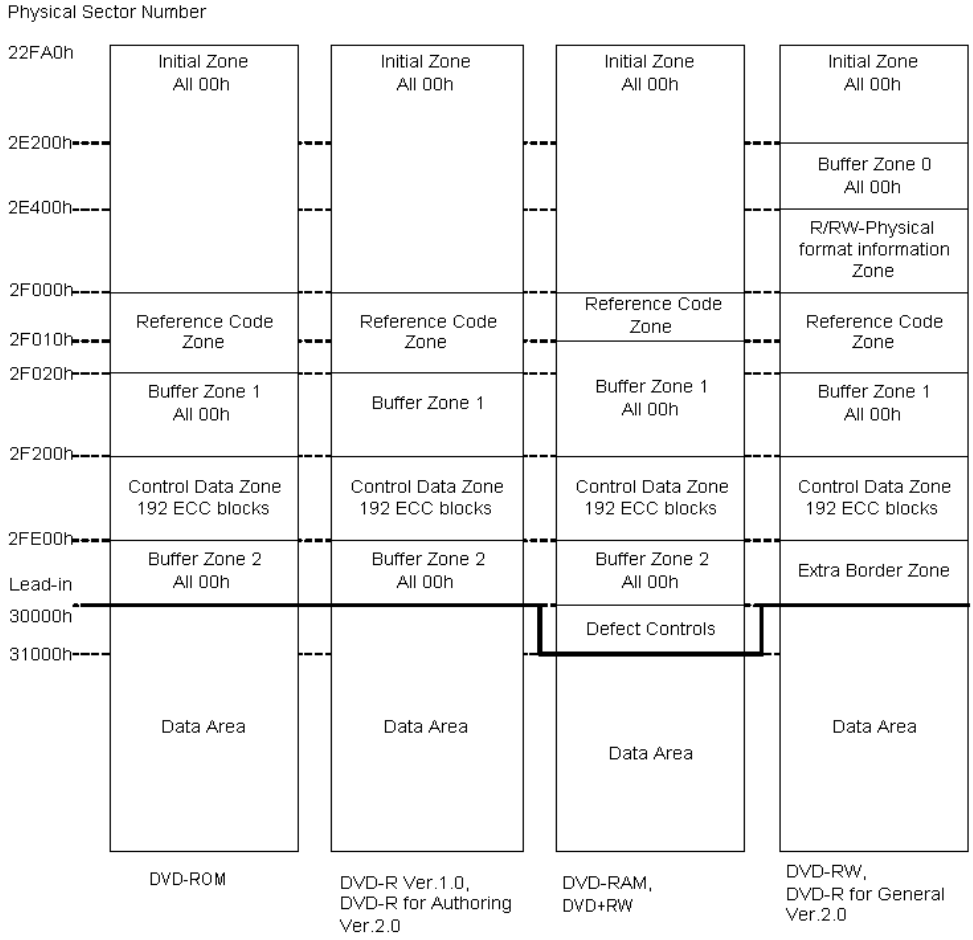

Рис.58. Форматы Lead-In областей DVD-дисков различных типов

Для всех типов дисков зона Control Data состоит из 192 ECC блоков. Все блоки этой зоны содержат одинаковую информацию. Структура ECC блока зоны Control Data представлена в таблице 10.

Таблица 10. Структура ECC блока зоны Control Data

Номер сектора	Описание
0	Информация о физическом формате диска, Physical Format Information (2048 bytes)
1	Информация о производителе диска, Disk Manufacturing Information (2048 bytes)
2 - 15	Зарезервированно (14 x 2048 bytes)

Формат поля Physical Format Information приведен в таблице 11.

Таблица 11. Формат поля Physical Format Information

Bit / Byte	7	6	5	4	3	2	1	0
0	Book Type				Part Version			

1	Disk Size			Maximum Rate
2	Reserved	Number of Layers	Track Path	Layer Type
3	Linear Density			Track Density
4-15	Data Area Allocation			
16	BCA Flag	Reserved		
17-2047	Medium Unique Data			

Поле Book Type определяет тип DVD-диска. Допустимые значения этого поля:
- 0000b – DVD-ROM
- 0001b – DVD-RAM
- 0010b – DVD-R
- 0011b – DVD-RW
- 1001b – DVD+RW
- 1010b – DVD+R

Остальные значения зарезервированы.

Поле Disk Size определяет диаметр DVD-диска, и может принимать следующие значения:
- 0000b – DVD-диск диаметром 12 см
- 0001b – DVD-диск диаметром 8 см

Остальные значения (0010b – 1111b) зарезервированы.

Поле Maximum Rate содержит значение максимальной скорости обмена данными с носителем:
- 0000b - 2.52 Mbps
- 0001b - 5.04 Mbps
- 0010b - 10.08 Mbps

Поле Track Path описывает взаимное направление треков информационных слоев, если поверхность носителя содержит более одного слоя. Содержание полей Data Area Allocation и Medium Unique Data зависят от типа DVD-диска.

11.4 R-Information Zone

На дисках типа DVD-R/-RW перед областью Information Zone расположена дополнительная зона - R-Information Zone. В ее состав входят две области:
- область калибровки мощности пишущего лазера (Power Calibration Area, PCA);
- область управления записью (Recording Management Area, RMA).

R-Information Zone занимает физические сектора с адресами 020800h – 02FFFFh для диска DVD-R Ver.1.0, и 01E800h – 203AFh для DVD-R for Authoring Ver.2.0 и DVD-R for General Ver.2.0. Структура DVD-R/-RW диска представлена на рис.59.

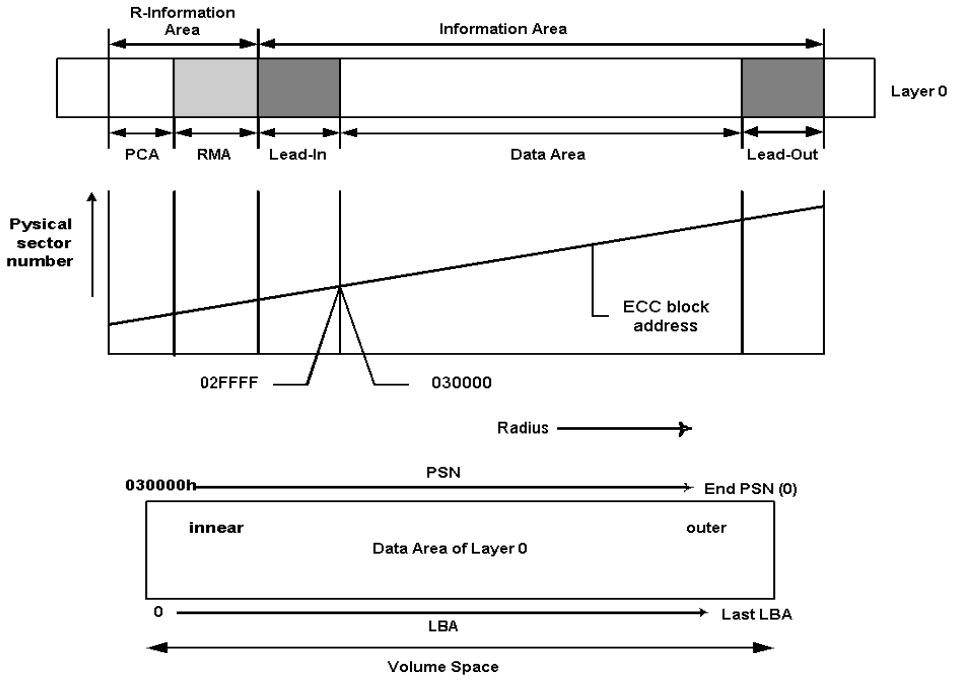

Рис.59. Физический и логический формат DVD-R/-RW диска

Область PCA занимает диапазон физических секторов с адресами от 020800h до 0223B0h и состоит из 7088 калибровочных участков. Структура этой области приведена на рис.60.

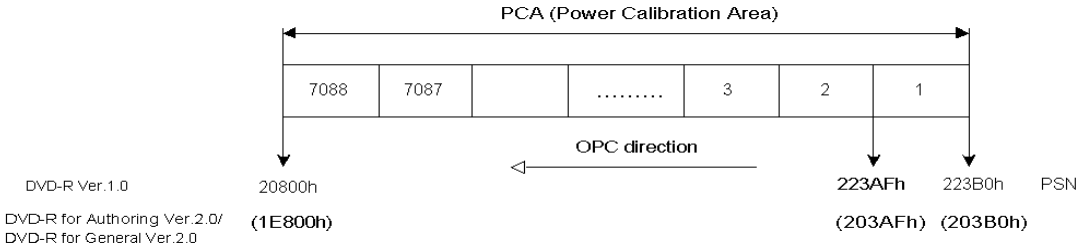

Рис.60. Структура PCA области DVD-R диска

Область RMA занимает диапазон физических секторов с адресами от 0223C0 до 024F80 и предназначена для хранения блоков данных управления записью (Recording Management Data, RMD). Прежде чем изучить структуру RMA области, необходимо определиться с таким понятием, как Border-область. С этой целью рассмотрим, какие режимы записи поддерживают DVD-R/-RW дисков.

11.5 Режимы записи DVD-R/-RW дисков

Запись на DVD-R диск выполняется только последовательно. Это означает, что хост не может записать данные в сектор, который выбран произвольным образом. Сектор для записи всегда заранее предопределен, и запись на чистый DVD-R диск всегда начинается с сектора, логический адрес которого равен нулю (LSN=0). Режим последовательной записи в спецификации обозначен как Sequential.

Запись и чтение данных с DVD-диска выполняется только целыми ECC блоками. Для чтения данных из одного сектора устройство считывает с диска целый ECC блок, в котором находится этот сектор, выполняет коррекцию ошибок, и затем извлекает из сектора данные. При записи устройство всегда выравнивает данные по границе ECC блока: если размер записываемых данных меньше размера ECC блока (данные не укладываются целиком в ECC блок), то в оставшиеся сектора этого блока записываются нулевые данные (рис.61).

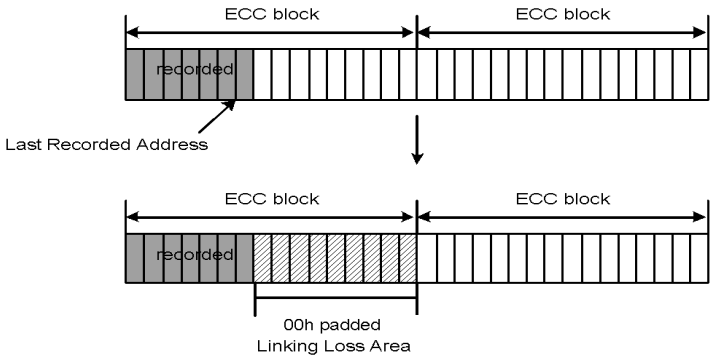

Рис.61. Выравнивание данных (padding) по границе ECC блока

Диски DVD-R поддерживают два типа записи - Disk-at-Once и Incremental. Для выбора режима служит поле Write Type страницы параметров режима записи (см. SCSI MMC-5 [1], табл. 606 «Write Parameters Page»). В режиме Disk-at-Once (DAO) запись данных на диск выполняется последовательно и без перерыва (рис.62). Добавить данные на диск, записанный в этом режиме, нельзя.

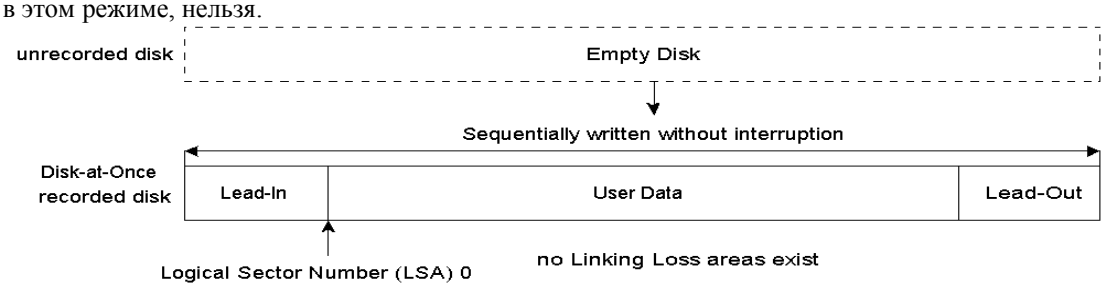

Рис.62. Запись DVD-R диска в режиме Disk-at-Once

Режим Incremental отличается от режима DAO тем, что позволяет добавлять (дописывать) данные на диск. При каждой остановке записи (например, по команде SYNCHRONIZE CACHE для выгрузки данных из внутреннего буфера устройства) на диске формируется специальная структура, которая называется областью потерь на соединение (связывание) – Linking Loss Area. Это своего рода аналог Pre-Gap и Post-Gap областей компакт-диска, предназначенный для разграничения различных участков записи.

Размер области Linking Loss Area может составлять 2 Kb и 32 Kb. Это значение показывает, сколько будет отдано секторов для соединение – один (2 Kb) или 16 (32 Kb, размер ECC блока). Сектора действительно теряются, т.к. полезных данных они не содержат. Для выбора размера соединительной области служит поле Link Size страницы параметров режима записи. Схема формирования области Linking Loss Area показана на рис.63.

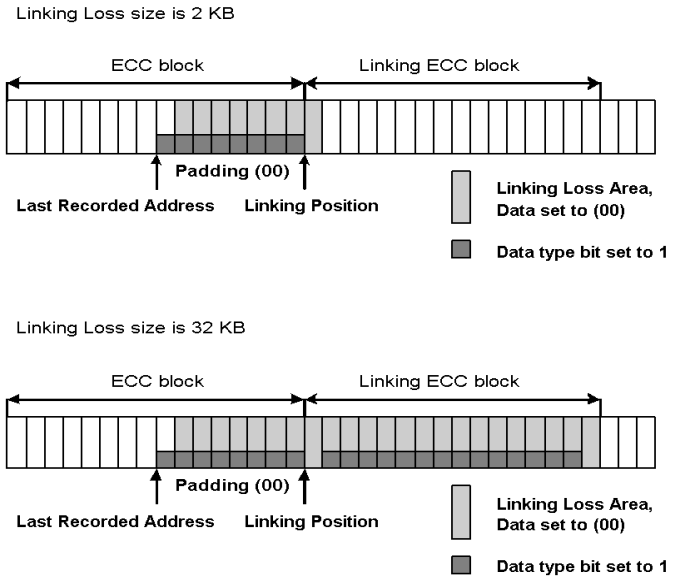

Рис.63. Схема формирования области Linking Loss Area

Все сектора, расположенные после последнего сектора, содержащего данные пользователя, используются для выравнивания данных по границе ECC блока. Поле MAIN DATA этих секторов содержит нулевые байты, и эти сектора входят в состав Linking Loss Area. Если размер Linking Loss Area равен 32 Kb, то все 16 секторов соединительного ECC блока используются для связывания. Если размер Linking Loss Area равен 2 Kb, то для связывания используется только первый сектор соединительного ECC блока, оставшиеся 15 секторов доступны для записи данных.

DVD-RW диски, кроме режимов Disk-at-once и Incremental, имеют еще один режим записи – режим ограниченной перезаписи, Restricted overwrite mode. В этом режиме запись может производится в произвольно выбранный сектор, но при этом объем записываемых данных и стартовый адрес для записи должны быть кратны 16 (размер ECC блока), в противном случае команда записи будет прервана с сообщением об ошибке 5/21/02 Invalid Adderss For Write. Подробнее об этом режиме мы поговорим в следующем разделе.

11.6 Border область

Если проводить сравнение с компакт-диском, то DVD-R/-RW диск, записанный в режиме DAO, можно рассматривать как аналог односессионного CD-диска, а в режиме Incremental - многосессионного. Если каждая сессия компакт-диска состоит из входной области Lead-In, выходной области Lead-Out и области данных Data Area, то DVD-диск имеет только одну Lead-In и Lead-Out область. При записи DVD-R/-RW диска в режиме Incremental в конце каждой сессии вместо Lead-Out формируется область, которая называется Border-Out. При добавлении (дописывании) новых данных на DVD-R/-RW диск вместо Lead-In области после последней области Border-Out записывается Border-In область. Совокупность Border-Out и Border-In областей называется Border Zone. Последовательность формирования Border Zone и структура Multiborder DVD-R/-RW диска показаны на рис.64 и 65.

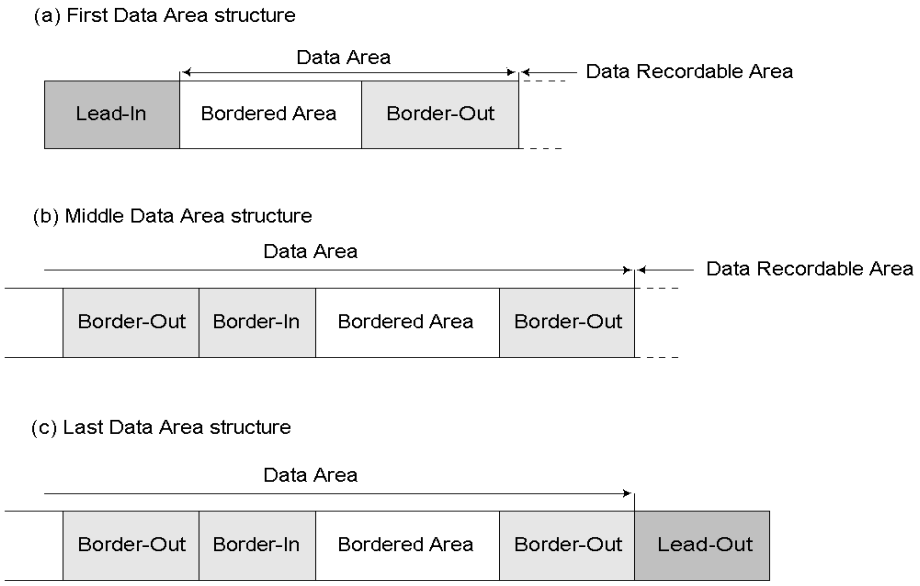

Рис.64. Последовательность формирования Border областей

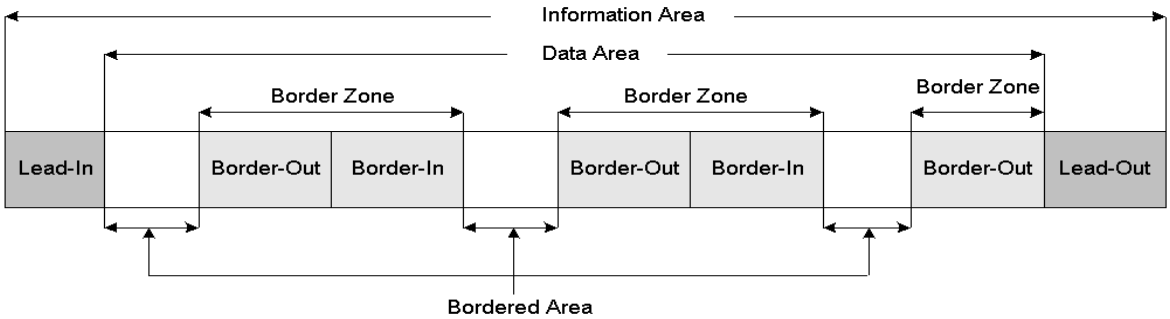

Рис.65. Структура Multiborder DVD-R/-RW диска, записанного в режиме Incremental

Адрес самой первой Border-Out область всегда расположен после PSN 3D700h для DVD-R Ver.1.0, и 3FF00h для DVD-R for Authoring Ver.2.0 и DVD-R for General Ver.2.0. Если последний сектор данных расположен ниже указанных PSN, то выполняется выравнивание данных по границе первой Border-Out области (3D700h и 3FF00h) путем заполнения оставшегося места нулевыми секторами.

Содержание Border-out и Border-in областей рассмотрим ниже, после того как ознакомимся со структурой RMA области DVD-R/-RW диска.

11.7 Структура RMA области DVD-R/-RW диска

11.7.1 Структура RMA области DVD-R диска

На рис.66 показана структура RMA области DVD-R диска.

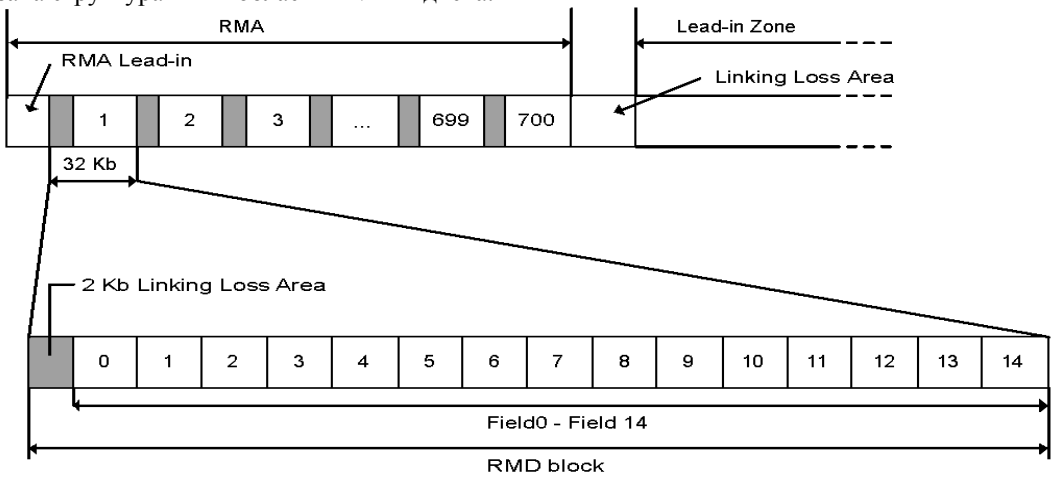

Рис.66. Структура RMA области DVD-R диска.

В самом начале расположена RMA Lead-in область размером в ECC блок. Состав RMA Lead-in:
- System Reserved Field (16 384 bytes);
- Unique ID Field (16 384 bytes).

Размер RMD блока – 32 Кбайт, т.е. один ECC блок. В состав блока входят 16 секторов - 15 секторов данных и один соединительный сектор. RMD блоки записываются в формате, который называется Format 1.

Поле Field 0 называется заголовком RMD блока (RMD Header). Это поле содержит общую информацию о диске и имеет следующий формат (табл.12):

Таблица 12. Формат поля Field 0 RMD блока (RMD Header)

Bit Byte	7	6	5	4	3	2	1	0
0-1	(MSB) RMD Format (LSB)							
2	Disk Status							
3	Reserved							
4-21	(MSB) Unique Disk Identifier (LSB)							
22-85	(MSB) Copy of Pre-recorded Information (LSB)							
86-2047	Reserved							

Значение поля RMD Format определяет формат полей Field 1~14. Для DVD-R диска значение этого поля всегда равно 1, и все поля RMD блока (Field 1~14) записываются в формате Format 1.

Поле Disk Status определяет состояние диска:
0 – область Data Area не содержит данных;
1 – диск записан в режиме Disk-at-once
2 – диск записан в режиме Incremental
3 – диск записан в режиме Incremental, но добавить на него данные нельзя (сессия закрыта)

В поле Field 1 содержится информацию о самом DVD-диске: производитель, серийный номер, значения OPC для различных скоростей и т.д. Формат этого поля приведен в таблице 45 *«RMD – Field 1 (logical unit & OPC information)»* спецификации INF-8090i [2].

Поле Field 3 содержит информацию о стартовых координатах Border-out областей диска и имеет следующий формат (табл.13):

Таблица 13. Формат поля Field 3 RMD блока (Format 1)

Bit / Byte	7	6	5	4	3	2	1	0
0-3	(MSB) Start Sector Number of Border-out #1 (LSB)							
4-7	(MSB) Start Sector Number of Border-out #2 (LSB)							
8-11	(MSB) Start Sector Number of Border-out #3 (LSB)							
..	..							
2036-2039	(MSB) Start Sector Number of Border-out #510 (LSB)							
2040-2043	(MSB) Start Sector Number of Border-out #511 (LSB)							
2044-2047	(MSB) Start Sector Number of Border-out #512 (LSB)							

В поле Field 4 находится информация о расположении RZone (треков) на DVD-диске. Поле имеет следующий формат (табл.6):

Таблица 6. Формат поля Field 4 RMD блока (Format 1)

Bit / Byte	7	6	5	4	3	2	1	0
0-1	(MSB) Invisible/Incomplete RZone Number (Last RZone Number) (LSB)							
2-3	(MSB) Current Appendable Reserved RZone Number 1 (LSB)							
4-5	(MSB) Current Appendable Reserved RZone Number 2 (LSB)							
6-15	Reserved							
16-19	(MSB) Start Sector Number of RZone #1 (LSB)							
20-23	(MSB) Last Recorded Address of RZone #1 (LSB)							
24-27	(MSB) Start Sector Number of RZone #2 (LSB)							
28-31	(MSB) Last Recorded Address of RZone #2 (LSB)							
..	..							
2032-2035	(MSB) Start Sector Number of RZone #253 (LSB)							
2036-2039	(MSB) Last Recorded Address of RZone #253 (LSB)							
2040-2043	(MSB) Start Sector Number of RZone #254 (LSB)							
2044-2047	(MSB) Last Recorded Address of RZone #254 (LSB)							

Поле Invisible/Incomplete RZone Number (Last RZone Number) содержит номер невидимой/незавершенной RZone. Если все RZone завершены, то в этом поле находится номер последней завершенной RZone.

Поле Start Sector Number of RZone #N содержит стартовый адрес RZone #N, а поле Last Recorded Address of RZone #N - адрес последнего записанного сектора RZone #N.

В полях Field 5 – Field 12 хранится информация о расположении RZone DVD-диска, в продолжение той, которая находится в поле Field4. DVD-диск может содержать 2302 RZone максимум.

Таблица 14. Формат полей Field 5 - 12 RMD блока (Format 1)

Bit / Byte	7	6	5	4	3	2	1	0
0-3	(MSB) Start Sector Number of RZone #n (LSB)							
4-7	(MSB) Last Recorded Address of RZone #n (LSB)							
8-11	(MSB) Start Sector Number of RZone #(n+1) (LSB)							
12-15	(MSB) Last Recorded Address of RZone #(n+1) (LSB)							
..								
2032-2035	(MSB) Start Sector Number of RZone #(n+253) (LSB)							
2036-2039	(MSB) Last Recorded Address of RZone #(n+253) (LSB)							
2040-2043	(MSB) Start Sector Number of RZone #(n+254) (LSB)							
2044-2047	(MSB) Last Recorded Address of RZone #(n+254) (LSB)							

11.7.2 Структура RMA области DVD-RW диска

Структура RMA области DVD-RW диска отличается от структуры RMA области DVD-R диска. RMD блоки DVD-RW диска могут быть записаны в трех форматах: Format 1, Format 2 и Format 3. RMD блоки формата Format 1 используются только в режиме последовательной записи (DAO/Incremental), блоки формата Format 3 – только в режиме ограниченной перезаписи, Restricted overwrite mode. RMD блоки формата Format 2 используются во всех режимах. Структура RMA области DVD-RW диска, записанного в последовательном режиме, показана на рис.67.

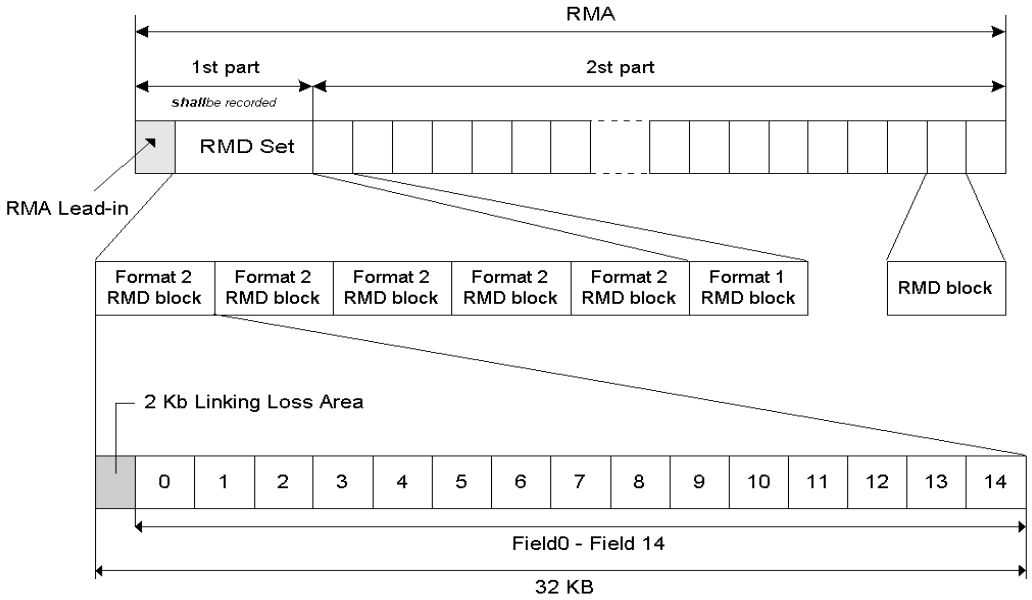

Рис.67. Структура RMA области DVD-RW диска, режим последовательной записи

Логически RMA область DVD-RW диска, записанного в последовательном режиме, разделена на две части. В первой части (1st part) находится RMD Set - набор из пяти RMD блоков формата Format 2, и все пять блоков содержат одинаковые данные. Первая часть используется, главным образом, для хранения информации о выполненных операциях стирания данных, в частности, тип операции стирания (полная или минимальная очистка диска, очистка RZone и т.п.), стартовый и конечный адреса участка, с которого стиралась информация и т.д.

Во второй части (2nd part) RMA области находятся 695 RMD блоков формата Format 1. Эти блоки, как и в случае DVD-R диска, содержат информацию о координатах Border областей и RZone диска. Структура полей RMD блока формата Format 1 DVD-RW и DVD-R дисков совпадают.

11.7.3 Структура полей RMD блоков DVD-RW диска

11.7.3.1 Содержание поля Field 0 RMD блока

Поле Field 0 является заголовком RMD блока (RMD Header). Это поле используется блоками всех форматов, и содержит общую информацию о диске. Формат этого поля приведен в таблице 15.

Таблица 15. Формат поля Field 0 RMD блока (RMD Header)

Bit Byte	7	6	5	4	3	2	1	0
0-1	(MSB) RMD Format (LSB)							
2	Disk Status							
3	Reserved							
4-21	(MSB) Unique Disk Identifier (LSB)							
22-85	(MSB) Copy of Pre-pit Information (LSB)							
86-127	Reserved							
128	RBG Information							
129-2047	Reserved							

RMD Format определяет формат всех следующих за Field 0 полей (Field 1~14), допустимые значения приведены в таблице 16.

Таблица 16. Значения поля RMD Format

0	Зарезервировано
1	Поля Field 1~14 записаны в формате Format 1 RMD format
2	Поля Field 1~14 записаны в формате Format 2 RMD format
3	Поля Field 1~14 записаны в формате Format 3 RMD format
4	Зарезервировано

Disk Status определяет состояние диска, значения этого поля приведены в таблице 17.

Таблица 17. Значения поля Disk Status

Величина	Назначение	Какими форматами используется
00h	Если поле Disk Status находится в RMD блоке формата Format 2, то это означает, что DVD диск записан в режиме Sequential, и текущее состояние диска определяет поле Disk Status из RMD блока формата Format 1. Для форматов Format 1 и 2 это означает отсутствие данных в области Data Recordable	Все форматы
01h	Диск записан в режиме Disk-at-once.	Format 1
02h	Диск записан в режиме Incremental	Format 1
03h	Диск записан в режиме Incremental и финализирован. Добавить данные нельзя	Format 1
04h	Была выполнена минимальная очистка диска	Format 1
05h	Выполняется очистка диска	Format 1
06h~0Fh	Зарезервировано	-
10h	Диск записан в режиме Restricted overwrite. Текущее состояние определяет поле Disk Status формата Format 3	Format 2
11h	Выполняется операция форматирования диска	Format 1, 3
12h	Диск в режиме Restricted overwrite	Format 3
13h	Последняя Border область находится в промежуточном состоянии (Intermediate state, см. п. 4.17.4.4 спецификации [2])	Format 3
14h~7Fh	Зарезервировано	-

11.7.3.2 Содержание полей Field 1~14 Format 1 RMD Set

Содержание полей Field 1~14 RMD блока формата Format 1 DVD-RW диска совпадает с одноименными полями RMD блока DVD-R диска (см. п. 11.7.1 «Структура RMA области DVD-R диска»).

11.7.3.3 Содержание полей Field 1~14 Format 2 RMD Set

Поле Field 1 содержит ссылку на Format 3 RMD Set текущего RMA сегмента. Формат Field 1 приведен в таблице 18.

Таблица 18. Format 2 RMD Field 1 (Pointer to Format 3 RMD Set)

Bit Byte	7	6	5	4	3	2	1	0
0-3	(MSB) Update Counter (LSB)							
4 – 7	(MSB) Format 3 RMD Set Pointer (LSB)							
8 - 11	Reserved							
12 – 13	(MSB) Erase Operation Counter (LSB)							
14 - 15	Reserved							
16 – 19	RSDS #2 ~ #28							
20-2047	Reserved							

Поле Update Counter содержит число перезаписей данного RMD Set. Начальное значение этого поля – 0, и при каждой перезаписи оно увеличивается на 1.

Format 3 RMD Set Pointer содержит ссылку на последний записанный Format 2 RMD Set текущего RMA сегмента. Эта ссылка представляет собой физический адрес стартового сектора Format 3 RMD Set.

Erase Operation Counter содержит количество выполненых операций стирания информации с диска. Начальное значение этого поля – 0, при каждой операции стирания оно увеличивается на 1. Детальная информация о выполненых операциях стирания находится в поле Field 2. Структура этого поля представлена в таблице 19.

Таблица 19. Format 2 RMD Field 2 (Erase Operation Information)

Bit Byte	7	6	5	4	3	2	1	0
0	Erase Operation Code (код операции стирания)							
1	Reserved							
2 – 5	(MSB) Erase Information 1 (LSB)							
6 – 9	(MSB) Erase Information 2 (LSB)							

19 – 2047	Reserved

Полный список кодов операции стирания информации находится в таблице 66 «Erase Operation Code and Erase Information fields definition» спецификации INF-8090i [2]. Мы рассмотрим только два значения:
- 1 – код операции полного стирания информации с диска;
- 2 – код операции минимальной очистки диска.

В обоих случаях поле Erase Information 1 содержит адрес в формате PSN первого сектора ECC блока, с которого началась операция стирания, поле Erase Information 2 - адрес в формате PSN последнего сектора ECC блока, на котором операция стирания информации завершилась.

Поля Field 3~14 Format 2 RMD Set зарезервированны и должны содержать нулевые значения.

11.8 Формат Border Zone

В состав Border Zone входят Border-in и Border-out области (рис.13 и 14). Формат Border Zone приведен на рис.68.

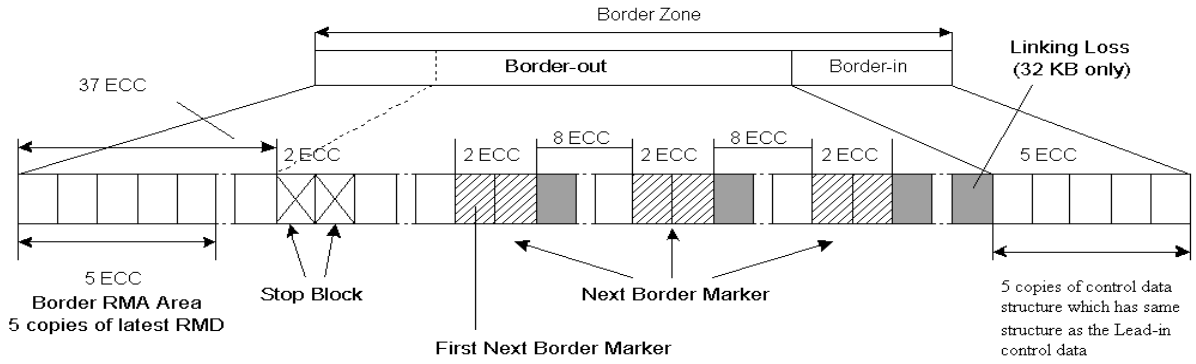

Рис.68. Формат Border Zone

Область Border-in содержит пять копий структуры данных Control Data, формат которой полностью совпадает с форматом зоны Control Data Zone, расположенной в Lead-In области диска (см. табл.11). Для предоставления информации о расположении Border Zone устройствам, которые не способны считывать RMA область, поле Physical Format Information структуры Control Data Border-in области содержит стартовые координаты текущей Border-out, следующей Border-in области, и физический номер последнего сектора области данных последней завершенной RZone.

В состав Border-out область входят область Border RMD Area размером 5 ECC блоков, два «стоповых» ECC блока Stop Blocks и три маркера Next Border Markers по два ECC блока каждый. В Border RMA Area каждый ECC блок содержит копию последнего записанного в RMA область RMD блока. Эта информация необходима устройствам, которые не способны считывать RMA область DVD-R/RW диска. «Стоповые» блоки Stop Blocks занимают 38-й и 39-й ECC блоки относительно начала Border-out и имеют атрибуты Lead-Out области. Маркеры Next Border Markers указывают на наличие/отсутствие следующей Border области. Детально содержание маркеров здесь не рассматривается, за подробностями обратитесь к спецификации.

12. Чтение служебных областей DVD-R/-RW диска

Чтение служебных областей DVD-R/RW диска позволяет получить информацию о физическом формате диска, режиме записи, логической структуре - стартовых координатах RZone, Border-областей и т.д. Для чтения служебных структур используется команда READ DVD STRUCTURE следующего формата:

READ DVD STRUCTURE CDB

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation Code (ADh)							
1	Reserved				Sub-command = 0000b			
2 - 5	(MSB) Address (LSB)							
6	Layer Number							
7	Format							
8 - 9	(MSB) Allocation Length (LSB)							
10	AGID		Reserved					
11	Control							

Поле Format определяет тип запрашиваемой информации, значение поля Address зависит содержимого поля Format.

12.1 Чтение блока Physical Format Information DVD-R/-RW диска

Для чтения блока Physical Format Information из Lead-In области диска поле Format должно содержать значение 0x10 (см. таблицу 310 «*Format code definitions for READ DVD STRUCTURE command*» спецификации INF-8090i [2]), поле Address не используется. Если команда выполнена успешно, устройство выдает заголовок пакета данных размером 4 байта, и пакет данных Layer Descriptor следующего формата (рис.69):

READ DVD STRUCTURE Data Format (With Format field = 10h)

Bit Byte	7	6	5	4	3	2	1	0
0 – 1	(MSB) DVD STRUCTURE Data Length (LSB)							
2 - 3	Reserved							
DVD Lead-in Structure								
0-2047	Layer Descriptor							

Рис.69. Формат пакета данных, получаемый по команде READ DVD STRUCTURE

Layer Descriptor

Bit Byte	7	6	5	4	3	2	1	0
0	Book Type				Part Version			
1	Disc Size				Maximum Rate			
2	Reserved	Number of Layers		Track Path	Layer Type			
3	Linear Density				Track Density			
4	00h							
5 - 7	(MSB) Starting Physical Sector Number of Data Area (LSB)							
8	00h							
9 - 11	(MSB) End Physical Sector Number of Data Area (LSB)							
12	00h							
13 - 15	(MSB) End Physical Sector Number in Layer 0 (LSB)							
16	BCA	Reserved						
17-2047	Medium Unique Data							

Рис.70. Формат поля Layer Descriptor

Поле Layer Descriptor можно описать при помощи следующей структуры:

```
typedef struct {
   __u8 part_version            :4;
   __u8 book_type         :4;
   __u8 max_rate          :4;
   __u8 disk_size         :4;
   __u8 layer_type        :4;
   __u8 track_path        :1;
   __u8 num_of_layers     :2;
   __u8 rez1              :1;
   __u8 track_dnst        :4;
   __u8 linear_dnst       :4;
   __u32 start_psn_da;
   __u32 end_psn_da;
   __u32 end_sect_L0;
   __u8 rez2              :7;
   __u8 BCA               :1;
} __attribute__ ((packed)) layer_descr_t;
```

В поле Data Area Allocation (байты 5-15) содержатся номера физических секторов начала и окончания области данных DVD-диска (Data Area). Содержание поля Medium Unique Data зависит от типа DVD-диска: для дисков типа DVD-R/-RW for General Ver.2.0 в этом поле находится физический адрес зоны Extra Border Zone – значение 02FE10h, и значение 2FFA0h – координаты блока Physical format information из Extra Border Zone (см. табл.22 спецификации INF-8090i [2]).

Если поле Format содержит значение 0x00, то с диска будет прочитан блок Physical Format Information из последней Border-in области. В этом случае в поле Medium Unique Data будет находиться адрес текущей Border-out области (по смещению 32 байта от начала Layer Descriptor), и адрес следующей Border-in области (см. табл.21 спецификации INF-8090i [2]).

Рассмотрим функцию, выполняющую чтение блока Physical Format Information. Входные параметры функции – значение поля Format, которое определяет, из какой области будет считываться блок. Работоспособность всех программ была проверена для ОС Linux, ядро 2.4.31, использовались следующие модели приводов:
- ASUS DRW-1604P 1.09.
- _NEC DVD-RW ND-3520A.

```
int read_PFI(__u8 ffield)
{
   __u8 read_dvd_struct_cmd[12];
   __u16 alloc_size = 2052;  // размер запрашиваемых данных (4 байта заголовка + 2048 байт данных)
   __u8 buff[2052];  // буфер для пакета данных
   __u32 Start_DA = 0, End_DA = 0; // начало и конец области данных диска
   __u32 u_part = 0; // значение из поля Medium Unique Data

   layer_descr_t *ldesc = (void *)(buff + 4); // указатель на начало Layer Descriptor

   memset(buff, 0, 2052);
   memset(read_dvd_struct_cmd, 0, 12);

/* Формируем командный пакет */
   read_dvd_struct_cmd[0] = 0xAD;  // код команды
   read_dvd_struct_cmd[7] = ffield; // значение поля Format field

/* Размер запрашиваемых у устройства данных */
   read_dvd_struct_cmd[8] = *((__u8 *)&alloc_size + 1);
   read_dvd_struct_cmd[9] = *((__u8 *)&alloc_size);

/* Посылаем устройству команду при помощи SCSI Generic драйвера.
 * Порядок работы с ним рассмотрен в [6]
 */
   test_unit_ready();
   if(send_cmd(read_dvd_struct_cmd, 12, SG_DXFER_FROM_DEV, \
     buff, 2052, 20) < 0) return -1;

/* Отображаем полученные результаты */
   printf("\nFormat field: 0x%X\n", ffield);
   printf("Book type: 0x%.2X\n", ldesc->book_type);
   printf("Disk size: %d\n", ldesc->disk_size);

/* Стартовые и конечные адреса области данных */
   Start_DA = __swab32(ldesc->start_psn_da);
   End_DA = __swab32(ldesc->end_psn_da);

   printf("Starting PSN of Data Area:  0x%X (%u)\n", Start_DA, Start_DA);
   printf("End PSN of Data Area:  0x%X (%u)\n", End_DA, End_DA);

/* Считываем значение из поля Medium Unique Data, находящееся по смещению 32 байта
 * от начала Layer Descriptor. Если Format = 0x10, то будет считано значение 02FE10h,
 * если  Format = 0x00 - стартовый адрес текущей Border-out области
 */
   memcpy((void *)&u_part, buff + 4 + 32, 4);
   u_part = __swab32(u_part);
   if(ffield == 0) printf("Start PSN of Current Border-out: ");
   printf("%Xh (%u)\n", u_part, u_part);

/* Считываем значение из поля Medium Unique Data, находящееся по смещению 36 байта
 * от начала пакета данных. Если Format = 0x10, то будет считано значение 2FFA0h,
 * если  Format = 0x00 - стартовый адрес следующей Border-in области
 */
   u_part = 0;
   memcpy((void *)&u_part, buff + 4 + 36, 4);
   u_part = __swab32(u_part);
   if(ffield == 0) printf("Start PSN of the next Border-in: ");
   printf("%Xh (%u)\n", u_part, u_part);

   return 0;
}
```

Полный текст программы чтения блока Physical Format Information находится в файле SOURCE/DVD/READ_DVD_STRUCT/read_PFI.c.

Устанавливаем в привод чистый DVD-RW диск и запускаем программу на выполнение. Проанализируем результаты работы программы. Вначале программа считывает данные PFI из Lead-In области. Результаты чтения:

```
Format field: 0x10
Book type: 0x03
Disk size: 0
Starting PSN of Data Area:  0x30000 (196608)
End PSN of Data Area:  0x26127F (2495103)
2FE10h (196112)
2FFA0h (196512)
```

Теперь PFI из последней Border-In:

```
Format field: 0x0
Book type: 0x03
Disk size: 0
Starting PSN of Data Area:  0x30000 (196608)
End PSN of Data Area:  0x0 (0)
Start PSN of Current Border-out: 0h (0)
Start PSN of the next Border-in: 0h (0)
```

Итак, результаты работы программы показали, что в приводе находится DVD-RW диск диаметром 120 мм, содержащий один информационный слой. Емкость диска составляет 2298496 секторов, или 4 707 319 808 байт. Стартовые адреса Border областей равны нулю, т.к. на диск ничего не записано.

Для следующего эксперимента нам понадобится DVD-диск, записанный в режиме Incremental. Для создания такого диска можно воспользоваться утилитой iso2dvd_multi, текст которой находится в файле SOURCE/DVD/WRITE/MULTI/iso2dvd_multi.c. Следует учесть один момент – для записи DVD-RW диска в режиме Incremental необходимо предварительно выполнить его ПОЛНУЮ очистку. Эта операция достаточно длительная и занимает ~30 минут.

Записываем на DVD-RW диск в режиме Incremental один трек размером 13 631 488 байт (6656 секторов), и смотрим на результаты работы функции read_PFI при Format Field = 0x00:

```
Format field: 0x0
Book type: 0x03
Disk size: 0
Starting PSN of Data Area:  0x30000 (196608)
End PSN of Data Area:  0x319FF (203263)
Start PSN of Current Border-out: 3FF00h (261888)
Start PSN of the next Border-in: 0h (0)
```

Физический адрес последнего записанного сектора в области данных 203263. Всего в области данных записано 6656 секторов (203263 – 196608 + 1, отсчет секторов ведется с 0), и это соответствует размеру трека. Обратим внимание на значение стартового сектора текущей Border-out области – 3FF00h, что в пересчете на логические сектора составляет 65280. В итоге мы потеряли 65280 – 6655 = 58625 секторов, или 120 Мбайт дискового пространства.

Теперь установим в привод DVD-RW диск, на который в режиме Incremental записан трек размером 1 417 376 Кбайт, или 708688 секторов. Результат работы функции при Format = 0x00:

```
Format field: 0x0
Book type: 0x03
Disk size: 0
Starting PSN of Data Area:  0x30000 (196608)
End PSN of Data Area:  0xDD04F (905295)
Start PSN of Current Border-out: DD060h (905312)
Start PSN of the next Border-in: 0h (0)
```

В этом случае потери на соединение составили всего 17 секторов.

12.2 Чтение RMA области DVD-R/-RW диска

Операция чтения RMA области позволяет получить информацию о логической структуре DVD-диска: режиме записи данных (DAO/Incremental), расположении (координатах) RZone и Border областей, количестве записанных на диск RZone. Отметим, что эту информацию можно прочитать как из RMA, так и из последней Border-out области диска. Для чтения RMD блоков, записанных в RMA область, поле Format команды READ DVD STRUCTURE должно быть равно 0x0D. В поле Address записывается стартовый номер сектора RMA области, начиная с которого будет выполняться чтение, в поле Allocation Length – размер считываемых данных. За одно обращение к диску можно прочитать один RMD блок размером 32768 байт (16 секторов).

В ответ на команду устройство вернет блок данных следующего формата:

READ DVD STRUCTURE Data Format (With Format field = 0Dh)

Bit Byte	7	6	5	4	3	2	1	0
0-1	(MSB) DVD STRUCTURE Data Length (LSB)							
2 - 3	Reserved							
DVD-R/-RW Recording Management Data Structure								
0 - 3	(MSB) Last Recorded RMA Sector Number (LSB)							
4 - n	RMD bytes							

Поле Last Recorded RMA Sector Number содержит номер последнего записанного в RMA область сектора. Считанные данные находятся в поле RMD Bytes.

Давайте проследим, как изменяется содержимое RMA области DVD-RW диска при записи на диск данных в режиме Sequentinal. Для этого из Format 1 RMD будем считывать поля Field 0 для определения статуса диска, и Field 4, содержащее координаты первых 254 RZone (см. табл. 6). Для чтения полей Field 0 и Field 4 из Format 1 RMD необходимо определить смещение к ним в секторах относительно начала RMA. Это смещение определяется по формуле: Start_Sector = 16 x (5 + N) + F + 1, где 16 – это число секторов в одном RMD блоке, N – номер RMD блока, из которого поле Field будет считываться, F – номер поля (0 или 4). Чтобы понять, откуда взялась эта формула, посмотрим рис.71:

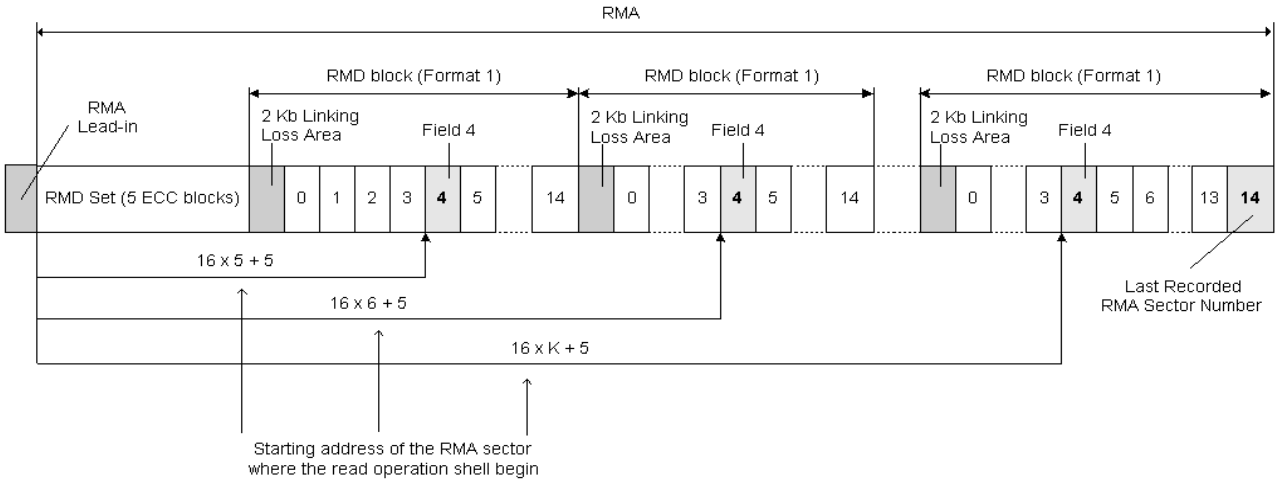

Рис.71. Порядок чтения RMD блоков из RMA области DVD-RW диска

Для чтения Field 4 необходимо пропустить 16 x 5 секторов, принадлежащих RMD Set (см. рис. 16), и затем сместится к нужному RMD блоку. Пятый сектор этого блока и будет искомым полем Field 4.

Чтение RMA области DVD-RW диска выполняет функция read_RMA:

```
int read_RMA()
{
#define SIZE 6152 // 2048 * 3 (3 сектора: Field 0, 1, 2) и 8 байт заголовка
#define SIZE1 2056 // 2048 (1 сектор) и 8  байт заголовка

    int i = 0, n = 0, k = 0;
    __u8 read_dvd_struct_cmd[12];
    __u8 buff[SIZE]; // буфер для данных
    __u8 *RMD = buff + 8; // указатель на начало RMD области
```

```
    __u8 disk_status = 0; // статус диска
    __u8 erase_code = 0;

    __u16 alloc_size = SIZE;
    __u16 EOC = 0; // Erase Operation Count
    __u16 RMD_Format = 0; // from  RMD Header (Field 0)
    __u16 Inv_RZone_num = 0; // Invisible RZone Number (Last Rzone Number)

    __u32 addr; // Поле Address командного пакета
    __u32 LRS = 0; // Last Recorded RMA sector number
    __u32 F3_RMD_Set_Pointer = 0; // Format 3 RMD Set Pointer
    __u32 Update_Counter = 0;
    __u32 EI1 = 0, EI2 = 0; // Erase Information 1/2
    __u32 Start_RZone = 0; // Start Sector Number of RZone #n
    __u32 LRA_RZone = 0; // Last Recorded Address of RZone #n

    memset(buff, 0, SIZE);

/* Из Format 2 RMD считываем поля Field #0 (RMD Header) для определения статуса диска и
 * формата остальных RMD блоков, Field #1 для опр. количества операций стирания и перезаписи,
 * Field #2 RMD для опр. кода операции стирания и участка с которого стиралась информация
 */
    memset(read_dvd_struct_cmd, 0, 12);
    read_dvd_struct_cmd[0] = 0xAD;
    read_dvd_struct_cmd[5] = 1; // адрес сектора для считывания
    read_dvd_struct_cmd[7] = 0x0D; // read RMA
    read_dvd_struct_cmd[8] = *((__u8 *)&alloc_size + 1);
    read_dvd_struct_cmd[9] = *((__u8 *)&alloc_size);

    test_unit_ready();
    if(send_cmd(read_dvd_struct_cmd, 12, SG_DXFER_FROM_DEV, \
     buff, alloc_size, 20) < 0) return -1;

 /* Определяем номер  последнего сектора, записанного в RMA */
    memcpy((void *)&LRS, buff + 4, 4);
    LRS = __swab32(LRS);
    printf("Last recorded RMA sector number: %u\n", LRS);

/* Field #0, первый сектор RMD блока. Отсчет секторов ведется с 0, 0-й сектор - это Linking Loss area */
    *((__u8 *)&RMD_Format) = RMD[1];
    *((__u8 *)&RMD_Format + 1) = RMD[0];

    disk_status = RMD[2];

    printf("Field 0 (RMD Header)\n");
    printf("RMD Format: %d\n", RMD_Format);
    printf("Disk Status: %.2Xh\n", disk_status);

 /* Field #1, 2-й сектор RMD блока */
    memcpy((void *)&Update_Counter, RMD + 0x800, 4);
    Update_Counter = __swab32(Update_Counter);

    memcpy((void *)&F3_RMD_Set_Pointer, (RMD + 4 + 0x800), 4);
    F3_RMD_Set_Pointer = __swab32(F3_RMD_Set_Pointer);

    *((__u8 *)&EOC) = *(RMD + 13 + 0x800);
    *((__u8 *)&EOC + 1) = *(RMD + 12 + 0x800);

    printf("Format 2 RMD Set Field 1\n");
    printf("Update Counter: %u\n", Update_Counter);
    printf("Format 3 RMD Set Pointer: %u\n", F3_RMD_Set_Pointer);
    printf("Erase Operation Counter: %d\n", EOC);

 /* Field #2, 3-й сектор RMD блока */
    erase_code  = RMD[0 + 0x1000];

    memcpy((void *)&EI1, RMD + 2 + 0x1000, 4);
    EI1 = __swab32(EI1);

    memcpy((void *)&EI2, RMD + 6 + 0x1000, 4);
```

```
    EI2 = __swab32(EI2);

    printf("Format 2 RMD Set Field 2\n");
    printf("Erase Operation Code: %d\n", erase_code);
    printf("Erase Information 1: %u\n", EI1);
    printf("Erase Information 2: %u\n\n\n", EI2);
}
```

/ Считываем информацию об RZone */*

```
    alloc_size = SIZE1; // размер данных для чтения – 1 сектор
    memset(buff, 0, SIZE);
    read_dvd_struct_cmd[8] = *((__u8 *)&alloc_size + 1);
    read_dvd_struct_cmd[9] = *((__u8 *)&alloc_size);
```

/ Пропускаем первые пять RMD блоков формата Format 2,*
** и считываем все Field #0 и Field #4 из каждого следующего RMD блока формата Format 1.*
** Field #0 - это 1й сектор от начала RMD блока, а Field #4 - 5й сектор от начала RMD блока*
** формата Format 1 (включая Linking Loss area)*
**/*

```
    for(k = 5;;k++) {

     addr = 16 * k + 1;
     if(addr > LRS) break;

     read_dvd_struct_cmd[2] = *((__u8 *)&addr + 3);
     read_dvd_struct_cmd[3] = *((__u8 *)&addr + 2);
     read_dvd_struct_cmd[4] = *((__u8 *)&addr + 1);
     read_dvd_struct_cmd[5] = *((__u8 *)&addr);

     test_unit_ready();
     if(send_cmd(read_dvd_struct_cmd, 12, SG_DXFER_FROM_DEV, \
                 buff, alloc_size, 200) < 0) return -1;

     *((__u8 *)&RMD_Format + 1) = RMD[0];
     *((__u8 *)&RMD_Format) = RMD[1];

     disk_status = RMD[2];

     printf("\nField #0 (sector %u)\n", addr);
     printf("RMD Format: %d\t", RMD_Format);
     printf("Disk Status: %.2X\n", disk_status);
```

/ Считываем Field #4 */*

```
     addr += 4;
     read_dvd_struct_cmd[2] = *((__u8 *)&addr + 3);
     read_dvd_struct_cmd[3] = *((__u8 *)&addr + 2);
     read_dvd_struct_cmd[4] = *((__u8 *)&addr + 1);
     read_dvd_struct_cmd[5] = *((__u8 *)&addr);

     test_unit_ready();
     if(send_cmd(read_dvd_struct_cmd, 12, SG_DXFER_FROM_DEV, \
                 buff, alloc_size, 200) < 0) return -1;

     printf("\nField #4 (sector %u)\n", addr);
     *((__u8 *)&Inv_RZone_num + 1) = RMD[0];
     *((__u8 *)&Inv_RZone_num) = RMD[1];
     printf("Invisible/Incomplete RZone Number: %d\n", Inv_RZone_num);

     printf("RZone\tSTART\tLAST\n");
     for(i = 1, n = 16; i <= Inv_RZone_num; i++, n += 8) {

         memcpy((void *)&Start_RZone, RMD + n, 4);
         Start_RZone = __swab32(Start_RZone);

         memcpy((void *)&LRA_RZone, RMD + n + 4, 4);
         LRA_RZone = __swab32(LRA_RZone);
         printf("%-d\t%-d\t%-d\n", i, Start_RZone, LRA_RZone);
     }
    }

    printf("\n");
```

```
    return 0;
}
```

Полный текст программы чтения RMA области DVD-RW диска находится в файле SOURCE/DVD/READ_DVD_STRUCT/read_RMA_DVDRW_SEQ.c.

Устанавливаем в привод полностью очищенный DVD-RW диск и запускаем программу на выполнение. Проанализируем результаты работы программы.

Вначале программа определяет номер последнего сектора, записанного в RMA область:

```
Last recorded RMA sector number: 111
```

Далее считывается Field 0 из первого блока RMD Set:

```
Field 0 (RMD Header)
    RMD Format: 2
    Disk Status: 00h
```

Значение RMD Format = 2 говорит о том, что поля Field 1~14 данного RMD записаны в формате Format 2. Диск записан в режиме Sequential, и текущее состояние диска определяет поле Disk Status из RMD блока формата Format 1 (см. таблицу 10).

Считывается Format 2 RMD Field 1:

```
Format 2 RMD Set Field 1
    Update Counter: 50
    Format 3 RMD Set Pointer: 0
    Erase Operation Counter: 17
```

Текущий RMD Set перезаписывался 50 раз. С диска 17 раз стиралась информация. RMD блоков формата Format 3 в RMA области нет, и соответствующий указатель установлен в 0.

Считывается Format 2 RMD Field 2:

```
Format 2 RMD Set Field 2
    Erase Operation Code: 1
    Erase Information 1: 132144
    Erase Information 2: 2495151
```

Диск подвергался полной очистке, код операции очистки равен 1. Адрес стартового сектора, с которого начиналась очистка информации, равен 132144. При полной очистке диска информация стирается из части RMA области и области данных пользователя + три ECC блока (см. [2], табл. 109 «*Blanking Types for DVD-RW*»). В RMA области нетронутыми остаются RMA Lead-in и шесть первых ECC блоков в составе RMD Set и одного Format 1 RMD блока. Зная, что стартовый адрес RMA области равен 132032 (203C0h, см. [2], рис.8), определим, с какого адреса будет выполняться операция полной очистки диска: 132032 + 16 * 7 = 132144.

Закончили чтение RMD Set, и переходим к чтению инфомации об RZone:

```
Field 0 (sector 81)
RMD Format: 1    Disk Status: 00

Field 4 (sector 85)
Invisible/Incomplete RZone Number: 1
RZone START      LAST
1    196608      0

Field 0 (sector 97)
RMD Format: 1    Disk Status: 00

Field 4 (sector 101)
Invisible/Incomplete RZone Number: 1
RZone START      LAST
1    196608      0
```

Тут все понятно – поле Disk Status равно 0, это значит что диск не содержит информации в области данных пользователя.

Теперь записываем на DVD-RW диск один трек размером 706240 Кбайт, или 353120 секторов. Режим записи - Incremental. Запускаем программу на выполнение и анализируем результаты.

```
Last recorded RMA sector number: 143
Field 0 (RMD Header)
    RMD Format: 2
    Disk Status: 00h
Format 2 RMD Set Field 1
    Update Counter: 50
    Format 3 RMD Set Pointer: 0
    Erase Operation Counter: 17
Format 2 RMD Set Field 2
    Erase Operation Code: 1
    Erase Information 1: 132144
    Erase Information 2: 2495151

Field #0 (sector 81)
RMD Format: 1    Disk Status: 00

Field #4 (sector 85)
Invisible/Incomplete RZone Number: 1
RZone START       LAST
1    196608      0

Field #0 (sector 97)
RMD Format: 1    Disk Status: 02

Field #4 (sector 101)
Invisible/Incomplete RZone Number: 2
RZone START       LAST
1    196608      0
2    549744      0

Field #0 (sector 113)
RMD Format: 1    Disk Status: 02

Field #4 (sector 117)
Invisible/Incomplete RZone Number: 2
RZone START       LAST
1    196608      549727
2    578416      0

Field #0 (sector 129)
RMD Format: 1    Disk Status: 02

Field #4 (sector 133)
Invisible/Incomplete RZone Number: 2
RZone START       LAST
1    196608      549727
2       578416    0
```

Хорошо видно, как меняется содержание RMA области при записи на диск новых данных. Самую «свежую» информацию содержит сектор, который находится по адресу (Last Recorded RMA Sector Number – 10), но это только для нашего случая, т.к. мы ограничили число RZone до 254 и не двигаемся дальше поля Field 4 RMD блока.

А теперь выполним минимальную очистку диска и снова прочитаем RMA.

```
Last recorded RMA sector number: 111
Field 0 (RMD Header)
    RMD Format: 2
    Disk Status: 00h
Format 2 RMD Set Field 1
    Update Counter: 51
    Format 3 RMD Set Pointer: 0
    Erase Operation Counter: 18
Format 2 RMD Set Field 2
    Erase Operation Code: 2
    Erase Information 1: 132144
    Erase Information 2: 196607
```

```
Field #0 (sector 81)
RMD Format: 1    Disk Status: 04

Field #4 (sector 85)
Invisible/Incomplete RZone Number: 1
RZone START      LAST
1    196608      0

Field #0 (sector 97)
RMD Format: 1    Disk Status: 04

Field #4 (sector 101)
Invisible/Incomplete RZone Number: 1
RZone START      LAST
1    196608      0
```

Изменились поля Update Counter, счетчик количества операций стирания и код операции стирания – все значения увеличились на 1. Поле Disk Status (Field #0) содержит значение 04 – была выполнена операция минимальной очистки диска (см. табл.10).

В отличии от DVD-RW, RMA область DVD-R диска не содержит набора RMD Set (рис.66), поэтому при чтении RMD нет необходимости смещаться на 5 ECC блоков вперед, и цикл чтения полей Field 4 RMD блоков примет следующий вид:

```
for(k = 0;;k++) {
  ...
  /* Тело цикла аналогично предыдущему примеру */
  ...
}
```

Копию последнего записанного в RMA область RMD блока (точнее, пять копий) содержит последняя Border-out область DVD-R/-RW диска, поэтому для получения информации о диске RMD блоки предпочтительнее считывать из этой области. Чтобы прочитать RMD блок из последней Border-out, поле Format команды READ DVD STRUCTURE должно быть равно 0x0C, в поле Address указывается номер поля Field, содержимое которого мы хотим прочитать. В ответ на команду устройство вернет блок данных следующего формата:

READ DVD STRUCTURE Data Format (With Format field = 0Ch)

Bit Byte	7	6	5	4	3	2	1	0
0 - 1	(MSB) DVD STRUCTURE Data Length (LSB)							
2 - 3	Reserved							
RMD in last Border-out								
0 .. n	RMD							

Чтение полей RMD блока из последней Border-out выполняет следующая функция:

```
int read_RMD()
{
/* Размер считываемых данных (в байтах) */
#define SIZE 2052 // 2048 bytes (1 sector)  + 4 header`s bytes

   int i, n = 16;
   __u8 read_dvd_struct_cmd[12];
   __u8 buff[SIZE]; // буфер для считываемых данных
   __u8 *RMD = buff + 4; // указатель на начало данных RMD блока
   __u8 disk_status = 0;

   __u16 alloc_size = SIZE;
   __u16 dvd_data_len = 0;
   __u16 Inv_RZone_num = 0; // номер последней невидимой/незавершенной RZone

   __u32 Start_BOUT = 0, Start_RZone = 0, LRA_RZone = 0;

/* Считываем Field #0 и проверяем статус диска */
   memset(buff, 0, SIZE);
   memset(read_dvd_struct_cmd, 0, 12);
   read_dvd_struct_cmd[0] = 0xAD;
```

```
    read_dvd_struct_cmd[7] = 0x0C; // read RMD in the last Border-out
    read_dvd_struct_cmd[5] = 0; // Address = 0, read Field #0
    read_dvd_struct_cmd[8] = *((__u8 *)&alloc_size + 1);
    read_dvd_struct_cmd[9] = *((__u8 *)&alloc_size);

    test_unit_ready();
    if(send_cmd(read_dvd_struct_cmd, 12, SG_DXFER_FROM_DEV, \
                buff, alloc_size, 20) < 0) return -1;

/* Размер считанных данных */
    *((__u8 *)&dvd_data_len) = buff[1];
    *((__u8 *)&dvd_data_len + 1) = buff[0];
    dvd_data_len += 2; // учитываем длину самого поля (2 bytes)
    printf("DVD STRUCTURE Data Length: %d\n", dvd_data_len);

/* Определяем статус диска */
    disk_status = *(RMD + 2);
    printf("Disk status: %d\n", disk_status);

/* Считываем Field #3, информацию о Border Zone */
    memset(buff, 0, SIZE);
    read_dvd_struct_cmd[5] = 3; // Address = 3, read Field #3

    test_unit_ready();
    if(send_cmd(read_dvd_struct_cmd, 12, SG_DXFER_FROM_DEV, \
                buff, alloc_size, 20) < 0) return -1;

/* Отображаем информацию о стартовых координатах Border-out областей диска */
    for(i = 0; i <= 2044; i += 4) {
      memcpy((void *)&Start_BOUT, RMD + i, 4);
      if(!Start_BOUT) break;
      Start_BOUT = __swab32(Start_BOUT);
      printf("Start Sector Number of Border-out #%d: %u\n", i/4 + 1, Start_BOUT);
    }

/* Считываем Field #4, информацию об RZone */
    memset(buff, 0, SIZE);
    read_dvd_struct_cmd[5] = 4; // Address = 4, read RMD #4

    test_unit_ready();
    if(send_cmd(read_dvd_struct_cmd, 12, SG_DXFER_FROM_DEV, \
                buff, alloc_size, 20) < 0) return -1;

/* Определяем номер невидимой/незавершенной RZone */
    *((__u8 *)&Inv_RZone_num + 1) = RMD[0];
    *((__u8 *)&Inv_RZone_num) = RMD[1];
    printf("Invisible/Incomplete RZone Number: %d\n", Inv_RZone_num);

/* Отображаем значения стартовой координаты и последнего записанного сектора данных RZone */
    printf("RZone\tSTART\tLAST\n");
    for(i = 1; i <= Inv_RZone_num; i++, n += 8) {

      memcpy((void *)&Start_RZone, RMD + n, 4);
      Start_RZone = __swab32(Start_RZone);

      memcpy((void *)&LRA_RZone, RMD + n + 4, 4);
      LRA_RZone = __swab32(LRA_RZone);

      printf("%-d\t%-d\t%-d\n", i, Start_RZone, LRA_RZone);
    }
    return 0;
}
```

Результат работы функции для DVD-RW диска, на котором записано два трека (размер первого трека – 718720 секторов, размер второго – 330992 секторов):

```
DVD STRUCTURE Data Length: 30724
Disk status: 2
Start Sector Number of Border-out #1: 915344
Start Sector Number of Border-out #2: 1284240
```

```
Invisible/Incomplete RZone Number: 3
RZone   START   LAST
1       196608  915327
2       953232  1284223
3       1291920 0
```

Значение статуса диска = 2, что соответствует режиму записи Incremental. Физический адрес стартового сектора первого трека – 196608, второго трека – 756624, третьего (невидимого) трека – 1095312. Для перевода в логический формат от каждого значения надо отнять 30000h. Информация о порядке определения стартовых адресах треков понадобится нам в следующем пункте, при рассмотрении режима записи Incremental.

Полный текст программы чтения RMD блоков из Border-out области находится в файле SOURCE/DVD/READ_DVD_STRUCT/read_RMD.c.

13. Запись DVD-R/-RW дисков в режиме Sequential

Записать информацию на DVD-диск ничуть не сложнее, чем на CD. В этом пункте мы рассмотрим, как выполнить запись данных на DVD в режимах Disk-at-once и Incremental. Под термином «данные» понимается файл в формате ISO, полученный при помощи утилиты mkisofs, или каким-либо другим способом.

13.1 Запись DVD-R/-RW дисков в режиме Disk-at-once

Алгоритм записи информации на DVD-R/RW в режиме DAO следующий:
- установливается требуемый режим записи – в поле Write Type страницы параметров режима записи записывается значение 2;
- командой RESERVE TRACK/RZONE резервируется для RZone пространство на диске;
- при помощи команды WRITE_10 выполняется запись данных на диск. Запись начинается с сектора с логическим адресом LSN = 0.
- по окончании передачи данных устройству послается команда SYNCHRONIZE CACHE. По этой команде все данные, находящиеся во внутреннем буфере устройства, будут перенесены на диск. Как только буфер опустеет, устройство сформирует на диске Lead-out область. Никаких дополнительных команд, типа CLOSE SESSION, для этого не надо.

Все функции, которые используются для реализации данного алгоритма, уже были рассмотрены нами в разделе о записи CD-дисков. Полный текст программы для записи DVD-R/-RW дисков в режиме DAO находится в файле ./ SOURCE/DVD/WRITE/DAO/iso2dvd_dao.c.

13.2 Запись DVD-R/-RW дисков в режиме Incremental

Как было отмечено в п.11.5, режим Incremental, в отличии от DAO, позволяет дописывать данные на диск. Для записи DVD-R/-RW диска в режиме Incremental в поле Write Type страницы параметров режима записи необходимо установить значение 3. Перед тем как записывать DVD-RW диск в режиме Incremental, необходимо выполнить его ПОЛНУЮ очистку.

Алгоритм записи Multiborder DVD-диска фактически не отличается от алгоритма записи многосессионного компакт-диска, см. п. 8.3.

При формировании ISO-образа (кроме первого), как и в случае многосессионного CD-диска, необходимо знать стартовый адрес последнего завершенного трека (RZone), и стартовый адрес невидимого трека (в терминологии CD – адрес следующей возможной области программ). Эти адреса можно извлечь из RMA области диска, из RMD блока последней Border-out, либо проанализировать информацию о треках на диске. Два первых способа мы только что рассмотрели в предыдущем пункте, поэтому остановимся на последнем.

Для получения информации о треке (RZone) используется команда READ TRACK INFORMATION. Формат этой команды приведен на рис.42. В ответ на эту команду устройство вернет блок информации о треке, структура которого показана на рис.43. Состояние трека определяют значения битов RT (Reserved Track), Blank, Packet/Inc и FP. Для невидимого трека биты принимают следующие значения (см. табл. 467 “Track Status Indications” спецификации SCSI MMC-5 [1]): RT = 0, Blank = 1, Packet/Inc = 1, FP = 0.

Рассмотрим пример. Имеется DVD-RW диск, на котором в режиме Incremental записан трека размером 349584 секторов. Считываем RMD блок из последней Border-out области. Результаты чтения:

```
Disk status: 2
```

```
Start Sector Number of Border-out #1: 546208
Invisible/Incomplete RZone Number: 2
RZone START       LAST
1     196608      546191
2     574880      0
```

Логический адрес стартового сектора невидимого трека – 378272.

Теперь считываем информацию о треках. Для этого используется программа, исходный текст которой расположен в файле ./SOURCE/DVD/READ_DISK_INFO/read_disk_info.c. Получаем следующий результат:

```
Track number: 1
Start address of track: 0
Next writable address: 0
Track size: 349584
Free blocks: 0
RT Blank PAC/INC FP: 0 0 1 0

Track number: 2
Start address of track: 378272
Next writable address: 378272
Track size: 1919616
Free blocks: 1919616
RT Blank PAC/INC FP: 0 1 1 0
```

Первый трек завершен, т.к. число свободных блоков равно 0. Второй трек является невидимым, и его стартовый адрес равен 378272.

Сформируем второй образ для записи на диск. Размер второго трека – 353120 секторов:

```
# mkisofs –R –J –l –C 0,378272 –M [имя файла устройства] -o track2.iso [входной файл]
```

Считываем RMD блок из последней Border-out области:

```
Disk status: 2
Start Sector Number of Border-out #1: 546208
Start Sector Number of Border-out #2: 928016
Invisible/Incomplete RZone Number: 3
RZone START       LAST
1     196608      546191
2     574880      927999
3     935696      0
```

Информация о треках:

```
Track number: 1
Start address of track: 0
Next writable address: 0
Track size: 349584
Free blocks: 0
RT Blank PAC/INC FP: 0 0 1 0

Track number: 2
Start address of track: 378272
Next writable address: 0
Track size: 353120
Free blocks: 0
RT Blank PAC/INC FP: 0 0 1 0

Track number: 3
Start address of track: 739088
Next writable address: 739088
Track size: 1558800
Free blocks: 1558800
RT Blank PAC/INC FP: 0 1 1 0
```

Исходный текст программы для записи DVD-R/-RW дисков в режиме Incremental находится в файле SOURCE/DVD/WRITE/MULTI/iso2dvd_multi.c.

В следующем выпуске книги будет рассмотрен режим записи DVD-RW диска Restricted overwrite.

14. Информационные ресурсы

1. Спецификация SCSI Multimedia Commands-5 (SCSI MMC-5), http://www.t10.org/ftp/t10/drafts/mmc5/mmc5r00.pdf.
2. Specification for ATAPI DVD Devices, ftp.seagate.com/sff/INF-8090.pdf
3. Information specification INF-8020i Rev 2.6. ATA Packet Interface for CD-ROMs SFF-8020i, http://www.stanford.edu/~csapuntz/specs/INF-8020.PDF.
4. SCSI-Generic-HOWTO, http://www.linux.org/docs/ldp/howto/SCSI-Generic-HOWTO/index.html.
5. Кулаков В. Програмирование дисковых подсистем. - СПб.: Питер, 2002. - 768 с.:ил.
6. Гук М. Аппаратные средства IBM PC. Энциклопедия, 2-е изд. - СПб.: Питер, 2003. - 923 с.:ил.
7. Гук М. Дисковая подсистема ПК. - СПб.: Питер, 2001.
8. Касперски Крис. Техника защиты компакт-дисков от копирования. - Спб.: БХВ-Петербург, 2004. - 464 с.:ил.
9. Касперски Крис. Статья "Способы взаимодействия с диском на секторном уровне", http://kpnc.opennet.ru/ATAPI.zip.
10. Introduction to CD and CD-ROM (with information on CD and CD-ROM formats, complete with diagrams and tables), http://www.disctronics.co.uk/downloads/tech_docs/cdintroduction.pdf.
11. Descriptions of mastering and replication of CD and DVD discs with diagrams, http://www.disctronics.co.uk/downloads/tech_docs/replication.pdf.
12. Comprising a comprehensive list of terms and words used in connection with CDs and DVDs and the applications that they support, http://www.disctronics.co.uk/downloads/tech_docs/glossary.pdf.
13. CD-Recordable FAQ, http://www.cdrfaq.org.
14. Standart ECMA-279. 80 mm (1,23 Gbytes per side) and 120 mm (3,95 Gbytes per side) DVD-Recordable Disk (DVD-R).
15. Standard ECMA-338. 80 mm (1,46 Gbytes per side) and 120 mm (4,70 Gbytes per side) DVD Re-recordable Disk (DVD-RW).